Karlsruher Institut für Technologie

Betriebssysteme
(Operating Systems)

Prof. Dr. Frank Bellosa
Lukas Werling, M.Sc.
Dr. Marc Rittinghaus

Nachname/ *Last name*	Vorname/ *First name*	Matrikelnr./ *Matriculation no*

Nachklausur
09.09.2021

- Bitte tragen Sie zuerst auf dem Deckblatt Ihren Namen, Ihren Vornamen und Ihre Matrikelnummer ein. Tragen Sie dann auf den anderen Blättern (auch auf Konzeptblättern) Ihre Matrikelnummer ein.

  Please fill in your last name, your first name, and your matriculation number on this page and fill in your matriculation number on all other (including draft) pages.

- Die Prüfung besteht aus 29 Blättern: Einem Deckblatt, 23 Aufgabenblättern mit insgesamt 5 Theorieaufgaben und 3 Programmieraufgaben sowie 5 Seiten Man-Pages.

  The examination consists of 29 pages: One cover sheet, 23 sheets containing 5 theory assignments as well as 3 programming assignments, and 5 sheets with man pages.

- Es sind keinerlei Hilfsmittel erlaubt!

  No additional material is allowed!

- Die Prüfung ist nicht bestanden, wenn Sie versuchen, aktiv oder passiv zu betrügen.

  You fail the examination if you try to cheat actively or passively.

- Sie können auch die Rückseite der Aufgabenblätter für Ihre Antworten verwenden. Wenn Sie zusätzliches Konzeptpapier benötigen, verständigen Sie bitte die Klausuraufsicht.

  You can use the back side of the assignment sheets for your answers. If you need additional draft paper, please notify one of the supervisors.

- Bitte machen Sie eindeutig klar, was Ihre endgültige Lösung zu den jeweiligen Teilaufgaben ist. Teilaufgaben mit mehreren widersprüchlichen Lösungen werden mit 0 Punkten bewertet.

  Make sure to clearly mark your final solution to each question. Questions with multiple, contradicting answers are void (0 points).

- Programmieraufgaben sind gemäß der Vorlesung in C zu lösen.

  Programming assignments have to be solved in C.

Die folgende Tabelle wird von uns ausgefüllt!
The following table is completed by us!

Aufgabe	T1	T2	T3	T4	T5	P1	P2	P3	Total
Max. Punkte	9	9	9	9	9	15	15	15	90
Erreichte Punkte									

Aufgabe T1: Grundlagen
Assignment T1: Basics

a) In einem herkömmlichen Betriebssystem findet eine Unterteilung der Ausführung in Benutzer- und Kernelmodus statt. Unter welchen Umständen kann es Sinn machen, auf eine solche Unterteilung zu verzichten? Geben Sie ein Beispiel an. **1 pt**

In a conventional operating system, the execution is separated into user and kernel mode. Under which circumstances can it make sense to abandon this separation? Give an example.

Nennen und begründen Sie einen Vor- und einen Nachteil einer gemeinsamen Ausführung von Anwendung und Betriebssystem im Kernelmodus. **2 pt**

Name and explain one advantage and one disadvantage of running the application and operating system together in kernel mode.

b) Welche Aufgabe kommt dem *System Call Dispatcher* zu? **1 pt**

What is the role of the system call dispatcher*?*

c) Multiprozessorsysteme können Interrupts üblicherweise an einen beliebigen Prozessor zustellen. Geben Sie zwei Gründe an, weshalb die Wahl des Zielprozessors die Performance des Systems beeinflussen kann. **2 pt**

Multi-processor systems are usually capable of delivering interrupts to an arbitrary processor. Give two reasons why the choice of the target processor may affect system performance.

d) Das `.bss`-Segment einer ELF-Datei wird vergrößert. Welche Auswirkung hat dies auf die Dateigröße sowie auf den Verbrauch von virtuellem und physischem Speicher? Gehen Sie von Demand Paging aus. Begründen Sie Ihre Antwort kurz. **3 pt**

The `.bss` segment of an ELF file is enlarged. How does this affect the file size and the consumption of virtual and physical memory? Assume demand paging. Briefly justify your answer.

Total: 9.0pt

Matrikelnummer/*Matriculation number*

Aufgabe T2: Prozesse und Threads

Assignment T2: Processes and Threads

a) Eine Kernfunktion des Betriebssystems ist die Prozessisolation. Was bedeutet das und wie wird diese Isolation realisiert? **1.5 pt**

A core functionality of the operating system is process isolation. What does that mean and how does the OS implement isolation?

Nennen Sie zwei Betriebssystemfunktionalitäten, durch die die Isolation zwischen Prozessen aufgeweicht wird. **1 pt**

Name two operating system functionalities that weaken the isolation between processes.

b) Was ist ein Daemon? Nennen Sie ein Beispiel. **1 pt**

What is a daemon? Give an example.

Was ist der *init-Prozess*? Welche Rolle spielt er im Bezug auf Daemons? **1 pt**

What is the init process? *Which role does it play with regard to daemons?*

c) Zum Starten eines neuen Threads muss Speicher alloziert werden. Welche Strukturen benötigen diesen Speicher? In welchem Adressraum befinden sich die Strukturen bei Einsatz des One-to-One-Modells? **2 pt**

When starting a new thread, memory needs to be allocated. Which structures need this memory? In the one-to-one model, in which address space are these structures located?

d) Welches Problem wird durch Virtual Round Robin im Vergleich zu normalem Round Robin gelöst? Erklären Sie Problem und Lösung. **2 pt**

Which problem does Virtual Round Robin solve in comparison with normal round robin? Explain the problem and the solution.

e) Nennen Sie einen Scheduling-Algorithmus, der nur auf reinen Batch-Systemen sinnvoll eingesetzt werden kann. **0.5 pt**

Name a scheduling algorithm which only makes sense on pure batch systems.

Total: 9.0pt

Matrikelnummer/*Matriculation number*

Aufgabe T3: Speicher

Assignment T3: Memory

a) Bei einem *Pufferüberlauf* schreibt ein Programm in Richtung ansteigender Adressen über die Grenzen eines Puffers. Liegt der Puffer auf dem Stack, kann die Rücksprungadresse überschrieben werden. Lässt sich dies verhindern, indem die Wachstumsrichtung des Stacks verändert wird? Begründen Sie Ihre Antwort. **1 pt**

During a buffer overflow *a program writes over the boundaries of a buffer towards increasing addresses. If the buffer is allocated on the stack, the return address can be overwritten. Can this be prevented by changing the direction of stack growth? Justify your answer.*

b) Diskutieren Sie Vor- und Nachteile von Bitmaps gegenüber einfach verketteter Listen (eine Seite pro Bit und Listenelement) zur Verwaltung freier Speicherseiten bzgl. Speicherverbrauch und algorithmischer Komplexität bei Allokation einer Seite. **2 pt**

Discuss the advantages and disadvantages of bitmaps versus singly-linked lists (one page per bit and list node) for managing free memory pages in terms of memory consumption and algorithmic complexity for allocating a page.

c) Speicher, der als interne Fragmentierung bezeichnet wird, darf nicht in der Frei-Liste auftauchen, wohingegen als externe Fragmentierung bezeichneter Speicher in der Frei-Liste auftauchen muss. **0.5 pt**

Memory designated as internal fragmentation must not appear in the free list, whereas memory designated as external fragmentation must appear in the free list.

☐ Ja / *Yes* ☐ Nein / *No*

d) Erläutern Sie den Begriff *Demand Paging*. Welchen Nachteil hat dieses Verfahren? **1 pt**

Explain the term demand paging. *What is the disadvantage of this method?*

e) Gegeben sei ein System mit einer 3-stufigen Seitentabelle mit 128-bit-Einträgen und 512 KiB-Seiten. Unterteilen Sie die folgende virtuelle Adresse, wie es für die Adressübersetzung nötig ist. Geben Sie dabei für jeden Abschnitt die Anzahl der Bits an. **2 pt**

Assume a system with a 3-level page table with 128-bit entries and 512 KiB pages. Divide the following virtual address as needed for the address translation. For each part, provide the number of bits.

Virtual Address

Wie groß ist der virtuelle Adressraum in Bytes? Zweierpotenz ausreichend. **0.5 pt**

What size is the virtual address space in bytes? Power of two sufficient.

Wie könnte die Speicherorganisation und Adressübersetzung bei gleichbleibender Adressraumgröße prinzipiell angepasst werden, um für dünn besetzte Adressräume Speicher zu sparen (keine Rechnung nötig)? **1 pt**

How could memory organization and address translation be adapted in principle to save memory for sparse address spaces (no calculation necessary)? The size of the address space should remain the same.

Wie wirkt sich dies auf die Effektivität des TLB aus? Begründen Sie Ihre Antwort. **1 pt**

What impact does this have on the effectiveness of the TLB? Justify your answer.

Total: 9.0pt

Aufgabe T4: Koordination und Kommunikation von Prozessen

Assignment T4: Process Coordination and Communication

a) Welche der in der Vorlesung vorgestellten Anforderungen an Synchronisierungsprimitive wird durch eine starke Semaphore erfüllt, durch eine schwache aber nicht? **0.5 pt**

Which of the requirements for synchronization primitives presented in the lecture is fulfilled by a strong semaphore, but not by a weak semaphore?

Welchen Thread weckt eine starke Semaphore auf, um die Anforderung zu erfüllen? **0.5 pt**

Which thread does a strong semaphore wake up to fulfill this requirement?

b) Sie möchten das vom Kernel angebotene Message Passing zwischen den User-Level-Threads eines Prozesses verwenden. Weshalb muss hierbei mindestens entweder Senden oder Empfangen asynchron sein? **1.5 pt**

You want to use the message passing provided by the kernel between the user-level threads of a process. In this case, why does at least either sending or receiving have to be asynchronous?

Weshalb eignet sich kernelseitiges direktes Message Passing nicht dazu, einzelnen User-Level-Threads gezielt Nachrichten zu schicken? **1 pt**

Why is it not possible to use kernel-based direct message passing to send messages specifically to individual user-level threads?

c) Nennen Sie die vier Bedingungen für einen Deadlock. **2 pt**

Name the four conditions for a deadlock.

Gegeben sei ein System, in dem Threads nie auf ein Spinlock zugreifen, falls sie bereits ein weiteres mit einer höheren virtuellen Adresse halten. Welche Bedingung für Deadlocks zwischen den Threads eines Prozesses wird dadurch verhindert? Erklären Sie kurz, weshalb die Bedingung nicht auftreten kann. Gehen Sie davon aus, dass nur Spinlocks zur Synchronisierung verwendet werden und dass physischer Speicher nicht mehrfach in den Adressraum abgebildet wird. **1.5 pt**

Assume a system where threads never access a spinlock if they already hold another spinlock with a higher virtual address. Which condition for deadlocks between the threads of a process is prevented by this scheme? Briefly explain why the condition is impossible. Assume that only spinlocks are used for synchronization and that physical memory is not mapped multiple times into the same address space.

Mehrere Prozesse werden durch Spinlocks in geteilten Speicherbereichen synchronisiert. Warum kann der Ansatz in manchen Situationen Deadlocks zwischen den Prozessen nicht verhindern? **2 pt**

Multiple processes are synchronized by spinlocks in shared memory regions. Why is the approach in some situations unable to prevent deadlocks between the processes?

Total: 9.0pt

Aufgabe T5: I/O, Hintergrundspeicher und Dateisysteme

Assignment T5: I/O, Secondary Storage, and File Systems

a) Auf Unix-System werden häufig wie unten dargestellt getrennte Dateisysteme für Wurzel- und Homeverzeichnis eingesetzt. Ein Benutzer versucht, mithilfe des `ln`-Kommandos Dateiverknüpfungen anzulegen.

Unix systems are often configured to use separate file systems for the root and home directories, as shown below. A user tries to create links with the `ln` command.

```
$ mount
[...]
/dev/sda1 on / type ext4 (rw)
/dev/sda2 on /home type ext4 (rw)

$ ln /etc/passwd /home/user/link1    # (1)
$ ln -s /etc/passwd /home/user/link2 # (2)
```

Welche Art von Verknüpfung versuchen die beiden Kommandos anzulegen? **1 pt**

What kind of link do the two commands try to create?

(1)

(2)

Sind die Befehle erfolgreich? Begründen Sie jeweils. Nehmen Sie an, dass die referenzierten Dateien und Ordner wie erwartet existieren. **2 pt**

Are the commands successful? Explain for both commands. Assume that the referenced files and directories exist as expected.

b) Trotz des Aufkommens von Solid State Disks (SSD) sind Festplatten heutzutage weiter relevant. Nennen Sie drei Vorteile von Festplatten. **1.5 pt**

Even with the rise of Solid State Disks (SSD), hard disks are still relevant today. Name three advantages of hard disks.

c) Nehmen Sie an, Sie betreuen eine Anwendung, die unter schlechter Performanz des Hintergrundspeichers leidet. Als Abhilfe erwägen Sie den Einsatz mehrerer Festplatten entweder als RAID 0-System oder als JBOD (Just a Bunch Of Disks), mit unabhängigen Dateisystemen pro Festplatte. Welche Variante würden Sie bevorzugen? Begründen Sie. **2 pt**

Suppose you are maintaining an application that is suffering from poor storage performance. As a solution, you consider using multiple hard disks either as a RAID 0 or as a JBOD (Just a Bunch Of Disks) with independent file systems per hard disk. Which variant would you prefer? Give reasons.

d) Ein Dateisystem nutzt Indexed Allocation und kann in einem Indexblock wahlweise 64 Pointer auf fixe Fragmente speichern oder 32 Extents. Welche zusätzliche Information muss in einem Extent vermerkt sein? **0.5 pt**

A file system uses indexed allocation and can store either 64 pointers to fixed fragments in an index block or 32 extents. Which additional piece of information is stored in an extent?

Nennen Sie zwei Vorteile davon, Extents einzusetzen. **1 pt**

Name two advantages of using extents.

e) Welche Aufgabe hat ein *fsck*-Programm? **1 pt**

What is the purpose of an fsck *program?*

Total: 9.0pt

Matrikelnummer/*Matriculation number*

Aufgabe P1: C Grundlagen
Assignment P1: C Basics

a) In dem untenstehenden Code haben sich 7 Fehler eingeschlichen. Markieren Sie die fehlerhaften Zeilen mit einem X und korrigieren Sie den Code. Gehen Sie von einem 64-Bit-System aus. **7 pt**

There are 7 errors in the code below. Mark the incorrect lines with an X and correct the code. Assume a 64-bit system.

```
struct entry {
    struct entry *next;
};

struct intentry {
    int32_t data;
    struct entry e;
};
```

```
☐ struct intentry *entry_to_intentry(struct entry *e) {
☐     return (struct intentry *) ((char *) e - 4);
☐ }
☐
☐ struct intentry *intentry_append(int32_t data, struct entry *e) {
☐     struct intentry *new = malloc(sizeof(*new));
☐     if (new) exit(1);
☐     new->data = data;
☐     new->e.next = e->next;
☐     e->next = &new;
☐     return new;
☐ }
☐
☐ void intentry_sum(struct entry *head) {
☐     int32_t sum;
☐     for (struct entry *e = head; e; e = e->next) {
☐         sum += entry_to_intentry(e)->data;
☐     }
☐     return sum;
☐ }
☐
☐ void intentry_free(struct entry *head) {
☐     struct entry *e = head->next;
☐     while (e) {
☐         struct intentry *ie = entry_to_intentry(e);
☐         free(ie);
☐         e = e->next;
☐     }
☐     head->next = NULL;
☐ }
```

```
/* example usage on next page */
```

Welche Datenstruktur implementiert der Code? **0.5 pt**

Which data structure does the code implement?

Was gibt das Programm (nach Korrektur aller Fehler) bei Ausführung der untenstehenden main-Funktion aus? **1 pt**

What does the program (after correcting all errors) print when executing the main function below?

```
int main() {
    struct entry head = {0};
    intentry_append(3, &head);
    intentry_append(2, &head);
    intentry_append(1, &head);
    printf("sum␣=␣%"PRId32"\n", intentry_sum(&head));
    intentry_free(&head);
}
```

b) Die Funktion `f()` des untenstehenden C-Programms wird zu der Assembly rechts kompiliert.

The function `f()` in the C program below compiles to the assembly on the right.

main.c

```
int f(int x, int y) {
  int sum = x + y;
  return sum;
}

int main() {
  return f(5, 7);
}
```

main.S (Ausschnitt / *excerpt*)

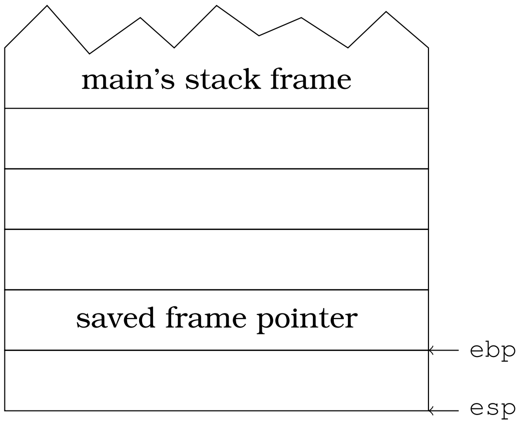
```
f(int, int):
        push    ebp
        mov     ebp, esp
        sub     esp, 4
        mov     edx, DWORD PTR [ebp+8]
        mov     eax, DWORD PTR [ebp+12]
        add     eax, edx
        mov     DWORD PTR [ebp-4], eax
        mov     eax, DWORD PTR [ebp-4]
        leave
        ret
```

Welche Aufrufkonvention kommt hier zum Einsatz? **0.5 pt**

Which calling convention is used here?

Füllen Sie in dem untenstehenden Diagramm den Inhalt des Stacks vor Ausführung der `leave`-Instruktion in `f()` aus. Geben Sie Zahlenwerte an wo möglich, ansonsten eine Beschreibung des Werts. **2 pt**

In the diagram below, fill in the contents of the stack before execution of the `leave` instruction in `f()`. Write numeric values if possible, otherwise give a description of the value.



c) Die Funktion `strncat()` konkateniert zwei Strings. Implementieren Sie die Funktion `my_strncat()` entsprechend der Beschreibung in der Manpage zu `strncat()`. Rufen Sie dabei keine Funktionen auf. **4 pt**

The function `strncat()` concatenates two strings. Without calling any functions, implement the function `my_strncat()` according to the manpage for `strncat()`.

```
char *my_strncat(char *dest, const char *src, size_t n) {
}
```

Total: 15.0pt

Aufgabe P2: Dateikomprimierung
Assignment P2: File Compression

Sie sollen ein Programm `compress` schreiben, das die Dateien, die durch die Kommandozeilenparameter spezifiziert werden, parallel einliest, komprimiert, und dann die Dateiinhalte durch die jeweiligen komprimierten Daten ersetzt. So soll zum Beispiel `compress A B` die Inhalte der Dateien `A` und `B` durch ihre komprimierte Form ersetzen.

- Sie müssen keine C-Header inkludieren.
- Sie müssen keine Fehlerbehandlung implementieren.
- Geben Sie jegliche im Code angeforderten Ressourcen wieder frei.

You shall write a program `compress` which reads the files specified by the command line parameters in parallel, compresses them, and then replaces the file contents with the corresponding compressed data. For example, `compress A B` replaces the contents of the files `A` and `B` by their compressed form.

- *You do not need to include any C headers.*
- *You do not have to implement any error handling.*
- *Free all resources allocated in the code.*

```
/* Global variables */
pthread_mutex_t mutex; /* mutex which protects "count" and "files" */
size_t count;          /* number of files yet to be processed */
char **files;          /* array of files yet to be processed */
                       /* ("files[0]" to "files[count-1]" are valid) */
```

a) Die folgende Funktion `replace_file()` ersetzt jegliche Inhalte der existierenden Datei `path` durch die Daten der Länge `size` an der Adresse `data`. Vervollständigen Sie den zweiten Parameter für `open()`, damit eine solche Ersetzung stattfindet. **0.5 pt**

The following function `replace_file()` replaces any content of the existing file `path` by the data of size `size` at the address `data`. Complete the second parameter of `open()` so that such a replacement occurs.

```
void replace_file(const char *path, const char *data, size_t size) {
    int fd = open(path,

.............................................................................................
    );
    write(fd, data, size);
    close(fd);
}
```

Vervollständigen Sie die Funktion `read_file()`, die die Inhalte der Datei `path` in den Speicher liest und einen Pointer auf die Daten zurückgibt. Die Länge der Daten soll an die durch `size` spezifizierte Adresse geschrieben werden. **5 pt**

- Bestimmen Sie vor dem Einlesen die Größe der Datei, um Speicher zu allozieren.
- Achtung: Der Speicher muss gegebenenfalls in einer der folgenden Teilaufgaben freigegeben werden.

Complete the function `read_file()` which reads the contents of the file `path` into memory and returns a pointer to the data. The size of the data shall be written to the address specified by `size`.

- *Determine the file size before reading the data to allocate memory.*
- *Caution: The memory may have to be released in one of the following sub-tasks.*

```
char *read_file(const char *path, size_t *size) {
    char *data; /* need to allocate the correct amount of memory! */
........................................................................................................

........................................................................................................

........................................................................................................

........................................................................................................

........................................................................................................

........................................................................................................

........................................................................................................

........................................................................................................

........................................................................................................

........................................................................................................

........................................................................................................

........................................................................................................

........................................................................................................

........................................................................................................

........................................................................................................

........................................................................................................
        return data;
}
```

b) Vervollständigen Sie die Funktion `compress_file()`, die eine einzige Datei (`path`) einliest, komprimiert und die Dateiinhalte mit den komprimierten Daten ersetzt. **3 pt**

- Verwenden Sie `compressBound()` und `compress()` aus der Zlib-Library zum Komprimieren. Hierfür finden Sie eine Manpage im Anhang der Klausur.
- Beachten Sie die Hinweise der Manpage zum Speicherbedarf der komprimierten Daten.

Complete the function `compress_file()` which reads a single file (`path`), compresses the data, and replaces the file contents with the compressed form.

- *For compression, use `compressBound()` and `compress()` from the Zlib library. A manpage for both can be found appended to the exam.*
- *Note the hints from the manpage with regards to memory usage for compressed data.*

```
void compress_file(const char *path) {
    size_t in_size;
    unsigned long out_size;
    char *input, *output;
    input = read_file(path, &in_size); /* read the file */

    /* compress the data: */
......................................................................................

......................................................................................

......................................................................................

......................................................................................

......................................................................................

......................................................................................
    replace_file(path, output, out_size); /* replace the file contents */
......................................................................................

......................................................................................

......................................................................................

......................................................................................
}
```

c) Das Programm soll mittels Threads parallelisiert werden, die jeweils die folgende Funktion `thread_entry()` ausführen. Vervollständigen Sie diese Funktion, die in einer Schleife jeweils einen Eintrag aus der durch `files` und `count` spezifizierten Liste entnimmt und die entsprechende Datei mittels `compress_file()` komprimiert. Dies soll wiederholt werden, solange noch Einträge vorhanden sind. **4 pt**

- Zugriffe auf die globalen Variablen müssen durch `mutex` geschützt werden. Komprimieren soll parallel erfolgen können.
- Sie können das erste Element aus der Liste entfernen, indem Sie die globale Variable `files` auf das zweite Element der Liste zeigen lassen.

The program shall be parallelized using threads which all call the following function `thread_entry()`. Complete this function which in a loop removes an entry from the list specified by `files` and `count` and compresses the corresponding file via `compress_file()`. This shall be repeated as long as entries remain.

- *Access to the global variables needs to be protected using `mutex`. Parallel compression shall be possible.*
- *You can remove the first element from the list by letting the global variable `files` point to the second element of the list.*

```
pthread_mutex_t mutex; /* mutex which protects "count" and "files" */
size_t count;          /* number of files yet to be processed */
char **files;          /* array of files yet to be processed */
                       /* ("files[0]" to "files[count-1]" are valid) */
void compress_file(const char *path);

void *thread_entry(void *param) {
    (void)param; /* unused */
















    return NULL;
}
```

d) Vervollständigen Sie die `main()`-Funktion des Programms, die eine feste Anzahl an Threads erstellt, die `thread_entry()` aufrufen, um die Dateien in `files` zu komprimieren. **2.5 pt**

Complete the `main()` function of the program which creates a fixed number of threads which call `thread_entry()` to compress the files specified by `files`.

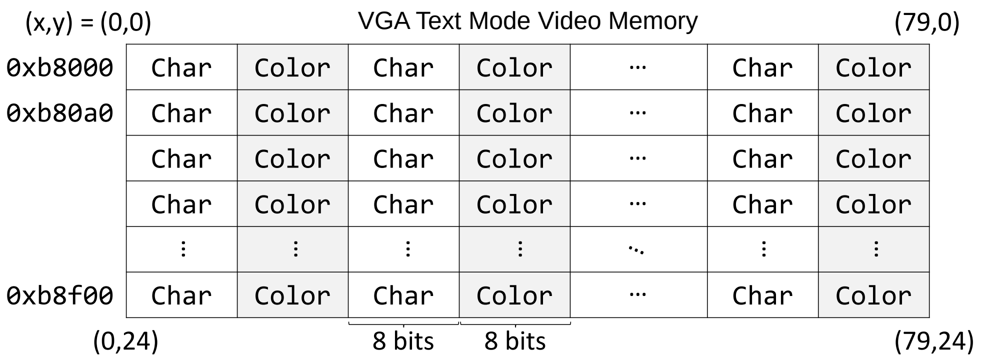
```c
pthread_mutex_t mutex; /* mutex which protects "count" and "files" */
size_t count;          /* number of files yet to be processed */
char **files;          /* array of files yet to be processed */
                       /* ("files[0]" to "files[count-1]" are valid) */
void *thread_entry(void *param);

#define NUM_THREADS 8
int main(int argc, char **argv) {
    int i;
    pthread_t threads[NUM_THREADS];

    /* initialization */
    files = argv + 1;
    count = argc - 1;
    pthread_mutex_init(&mutex, NULL);

    /* create threads */
    for (i = 0; i < NUM_THREADS; i++) {
.............................................................................................



.............................................................................................



.............................................................................................
    }

    /* wait for the threads to exit */
    for (i = 0; i < NUM_THREADS; i++) {
.............................................................................................



.............................................................................................



.............................................................................................
    }

    pthread_mutex_destroy(&mutex);
    return 0;
}
```

Total: 15.0pt

Aufgabe P3: VGA Terminal
Assignment P3: VGA Terminal

Der Video Graphics Array (VGA) Standard definiert eine einfache Schnittstelle, um Ausgaben auf dem Bildschirm zu erzeugen. VGA unterstützt dabei einen Textmodus mit 80×25 Zeichen (Breite×Höhe), bei dem ASCII-Zeichen angezeigt werden können, indem sie in einen vordefinierten Speicherbereich ab Adresse `0xb8000` geschrieben werden. Jedes Zeichen umfasst 16 Bits, wobei das erste Byte das ASCII-Zeichen und das zweite Byte eine Farbkodierung enthält (unabhängig von der Endianness der CPU).

The Video Graphics Array (VGA) standard defines a simple interface to produce output on the screen. VGA supports a text mode with 80×25 characters (width×height), which allows displaying ASCII characters by writing them to a predefined memory area starting at address `0xb8000`. Each character is 16 bits, where the first byte contains the ASCII code and the second byte contains a color code (independent of the endianness of the CPU).



```
#include <inttypes.h>

#define VGA_W    80         /* Screen width in characters              */
#define VGA_H    25         /* Screen height in characters             */

#define VGA_MEM  0xb8000    /* Base address of VGA memory              */

uint8_t term_fg, term_bg;   /* Current terminal fore-/background color */
uint8_t term_x,  term_y;    /* Current terminal cursor position        */

/* VGA word (character + color) declaration */





/* Pointer to VGA memory */
vga_word_t *vga = (vga_word_t*)(VGA_MEM);

/* VGA access macro */

#define VGA(x, y)
```

a) Ergänzen Sie auf Seite 20 einen Typ `vga_word_t`, der ein Zeichen im VGA-Speicher inkl. Farbcodierung repräsentiert (d. h. ein VGA-Wort). Der Typ soll unabhängig von der Endianness des Zielsystems sein. **1 pt**

On page 20, add a type `vga_word_t` which represents a character in VGA memory including its color code (i.e., one VGA word). The type shall be independent from the endianness of the target system.

b) Vervollständigen Sie die Funktion `vga_word()`, welche für ein ASCII-Zeichen (`c`) und eine Vorder- (`fg`) und Hintergrundfarbe (`bg`) ein `vga_word_t` zurückgibt. **2 pt**

- Die Farben werden im VGA-Speicher als 4-bit Indizes in eine VGA-Farbpalette angegeben. Die Vordergrundfarbe ist in den niederwertigen 4 Bits und die Hintergrundfarbe in den höherwertigen 4 Bits des `Color` Bytes gespeichert.
- Übernehmen Sie jeweils nur die niederwertigsten 4 Bits von `fg` und `bg` in das VGA-Wort.

Complete the function `vga_word()` which returns a `vga_word_t` for a given ASCII character (`c`), foreground color (`fg`), and background color (`bg`).

- *The colors are specified in VGA memory as 4-bit indices into a VGA color palette. The foreground color is stored in the least significant bits and the background color in the most significant bits of the `Color` byte.*
- *Transfer only the least significant 4 bits of `fg` and `bg` to the VGA word.*

```
vga_word_t vga_word(char c, uint8_t fg, uint8_t bg) {
.......................................................................................
.......................................................................................
.......................................................................................
.......................................................................................
.......................................................................................
.......................................................................................
.......................................................................................
.......................................................................................
.......................................................................................
}
```

c) Vervollständigen Sie das Makro `VGA(x, y)` auf Seite 20, welches mittels der Variablen `vga` das VGA-Wort an der Position (x,y) adressiert. Das Makro soll Lese- und Schreibzugriff der folgenden Form ermöglichen, wobei `x` und `y` beliebige Ausdrücke sein dürfen: **1 pt**

Complete the macro `VGA(x, y)` on page 20 which addresses the VGA word at position (x,y) using the variable `vga`. The macro shall allow read and write access of the following form, where `x` and `y` may be arbitrary expressions:

```
vga_word_t w;

w = VGA(x1, y1);
VGA(x2, y2) = w;
```

d) Vervollständigen Sie die Funktion `term_clear()`, welche den vollständigen VGA-Speicher mit Leerzeichen (`' '`) in der durch `fg` und `bg` angegebenen Farbkodierung überschreibt. **2.5 pt**

- Übernehmen Sie die Werte von `fg` und `bg` als neue Terminal-Farbkodierung.
- Setzen Sie die Position des Cursors auf (`0`,`0`) zurück.

Complete the function `term_clear()` which overwrites the entire VGA memory with spaces (`' '`) in the color code specified by `fg` and `bg`.

- *Take the values of `fg` and `bg` as new terminal color code.*
- *Reset the position of the cursor to (`0`,`0`).*

```
vga_word_t vga_word(char c, uint8_t fg, uint8_t bg);

void term_clear(uint8_t fg, uint8_t bg) {
.......................................................................................
.......................................................................................
.......................................................................................
.......................................................................................
.......................................................................................
.......................................................................................
.......................................................................................
.......................................................................................
.......................................................................................
.......................................................................................
.......................................................................................
.......................................................................................
.......................................................................................
.......................................................................................
.......................................................................................
.......................................................................................
}
```

e) Vervollständigen Sie die Funktion `term_scroll()`, welche den Inhalt des VGA-Speichers um eine Zeile nach oben rückt und dabei die oberste Zeile verwirft. **3 pt**

- Füllen Sie die neue Zeile mit Leerzeichen (`' '`) in der Terminal-Farbkodierung.
- Wenn möglich, rücken Sie den Cursor ebenfalls um eine Zeile nach oben.

Complete the function `term_scroll()` which moves the contents of the VGA memory up one line, discarding the top line.

- *Fill the new line with spaces (`' '`) in the terminal color code.*
- *If possible, also move the cursor up one line.*

```
vga_word_t vga_word(char c, uint8_t fg, uint8_t bg);

void term_scroll(void) {
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
}
```

f) Vervollständigen Sie die Funktion `term_putc()`, welche das ASCII-Zeichen in der Terminal-Farbkodierung an der aktuellen Position des Cursors ausgibt. **5.5 pt**

- Geben Sie nur druckbare ASCII-Zeichen aus (siehe ASCII-Tabelle).
- Implementieren Sie die Steuerzeichen für Wagenrücklauf ($\boxed{\text{CR}}$ - Cursor zum Zeilenanfang) und Zeilenvorschub ($\boxed{\text{LF}}$ - Cursor um eine Zeile nach unten).
- Am Zeilenende springt der Cursor an den Anfang der nächsten Zeile.
- Rufen Sie `term_scroll()` auf, wenn nötig.

Complete the function `term_putc()` which outputs the ASCII character in the terminal's color code at the current position of the cursor.

- *Only output printable ASCII characters (see ASCII table).*
- *Implement the control code for carriage return ($\boxed{\textit{CR}}$ - cursor to beginning of line) and line feed ($\boxed{\textit{LF}}$ - cursor down one line).*
- *At the end of the line the cursor jumps to the beginning of the next line.*
- *Call `term_scroll()` if necessary.*

```
vga_word_t vga_word(char c, uint8_t fg, uint8_t bg);
void term_scroll(void);

void term_putc(char c) {
......................................................................................
......................................................................................
......................................................................................
......................................................................................
......................................................................................
......................................................................................
......................................................................................
......................................................................................
......................................................................................
......................................................................................
......................................................................................
......................................................................................
......................................................................................
......................................................................................
......................................................................................
......................................................................................
......................................................................................
......................................................................................
......................................................................................
......................................................................................
......................................................................................
......................................................................................
......................................................................................
......................................................................................
}
```

Code	...0	...1	...2	...3	...4	...5	...6	...7	...8	...9	...A	...B	...C	...D	...E	...F
0...	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1...	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2...	␣	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3...	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4...	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5...	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6...	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7...	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Total: 15.0pt

NAME

open – open and possibly create a file

SYNOPSIS

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open(const char ***_pathname_, int** _flags_**);**
int open(const char ***_pathname_, int** _flags_**, mode_t** _mode_**);**

DESCRIPTION

The **open**() system call opens the file specified by _pathname_. If the specified file does not exist, it may optionally (if **O_CREAT** is specified in _flags_) be created by **open**().

The return value of **open**() is a file descriptor, a small, nonnegative integer that is used in subsequent system calls (**read**(2), **write**(2), **lseek**(2), **fcntl**(2), etc.) to refer to the open file. The file descriptor returned by a successful call will be the lowest-numbered file descriptor not currently open for the process.

A call to **open**() creates a new _open file description_, an entry in the system-wide table of open files. The open file description records the file offset and the file status flags (see below). A file descriptor is a reference to an open file description; this reference is unaffected if _pathname_ is subsequently removed or modified to refer to a different file. For further details on open file descriptions, see NOTES.

The argument _flags_ must include one of the following _access modes_: **O_RDONLY**, **O_WRONLY**, or **O_RDWR**. These request opening the file read-only, write-only, or read/write, respectively.

In addition, zero or more file creation flags and file status flags can be bitwise-_or_'d in _flags_. The _file creation flags_ are **O_CLOEXEC**, **O_CREAT**, **O_DIRECTORY**, **O_EXCL**, **O_NOCTTY**, **O_NOFOLLOW**, **O_TMPFILE**, and **O_TRUNC**. The _file status flags_ are all of the remaining flags listed below. The distinction between these two groups of flags is that the file creation flags affect the semantics of the open operation itself, while the file status flags affect the semantics of subsequent I/O operations. The file status flags can be retrieved and (in some cases) modified; see **fcntl**(2) for details.

The abridged list of file creation flags and file status flags is as follows:

O_APPEND
The file is opened in append mode. Before each **write**(2), the file offset is positioned at the end of the file, as if with **lseek**(2). The modification of the file offset and the write operation are performed as a single atomic step.

O_APPEND may lead to corrupted files on NFS filesystems if more than one process appends data to a file at once. This is because NFS does not support appending to a file, so the client kernel has to simulate it, which can't be done without a race condition.

O_CLOEXEC (since Linux 2.6.23)
Enable the close-on-exec flag for the new file descriptor. Specifying this flag permits a program to avoid additional **fcntl**(2) **F_SETFD** operations to set the **FD_CLOEXEC** flag.

Note that the use of this flag is essential in some multithreaded programs, because using a separate **fcntl**(2) **F_SETFD** operation to set the **FD_CLOEXEC** flag does not suffice to avoid race conditions where one thread opens a file descriptor and attempts to set its close-on-exec flag using **fcntl**(2) at the same time as another thread does a **fork**(2) plus **execve**(2). Depending on the order of execution, the race may lead to the file descriptor returned by **open**() being unintentionally leaked to the program executed by the child process created by **fork**(2). (This kind of race is in principle possible for any system call that creates a file descriptor whose close-on-exec flag should be set, and various other Linux system calls provide an equivalent of the **O_CLOEXEC** flag to deal with this problem.)

O_CREAT
If _pathname_ does not exist, create it as a regular file.

The owner (user ID) of the new file is set to the effective user ID of the process.

The _mode_ argument specifies the file mode bits be applied when a new file is created. This argument must be supplied when **O_CREAT** or **O_TMPFILE** is specified in _flags_; if neither **O_CREAT** nor **O_TMPFILE** is specified, then _mode_ is ignored. The effective mode is

modified by the process's _umask_ in the usual way: in the absence of a default ACL, the mode of the created file is _(mode & ˜umask)_. Note that this mode applies only to future accesses of the newly created file; the **open**() call that creates a read-only file may well return a read/write file descriptor.

The following symbolic constants are provided for _mode_:

S_IRWXU
00700 user (file owner) has read, write, and execute permission

S_IRUSR
00400 user has read permission

S_IWUSR
00200 user has write permission

S_IXUSR
00100 user has execute permission

S_IRWXG
00070 group has read, write, and execute permission

S_IRGRP
00040 group has read permission

S_IWGRP
00020 group has write permission

S_IXGRP
00010 group has execute permission

S_IRWXO
00007 others have read, write, and execute permission

S_IROTH
00004 others have read permission

S_IWOTH
00002 others have write permission

S_IXOTH
00001 others have execute permission

According to POSIX, the effect when other bits are set in _mode_ is unspecified.

O_DIRECTORY
If _pathname_ is not a directory, cause the open to fail. This flag was added in kernel version 2.1.126, to avoid denial-of-service problems if **opendir**(3) is called on a FIFO or tape device.

O_TRUNC
If the file already exists and is a regular file and the access mode allows writing (i.e., is **O_RDWR** or **O_WRONLY**) it will be truncated to length 0. If the file is a FIFO or terminal device file, the **O_TRUNC** flag is ignored. Otherwise, the effect of **O_TRUNC** is unspecified.

RETURN VALUE

open() returns the new file descriptor, or −1 if an error occurred (in which case, _errno_ is set appropriately).

NAME

compress, compressBound – zlib general purpose compression library

SYNOPSIS

#include <zlib>

int compress(unsigned char **dest*, **unsigned long** **destLen*,
    const unsigned char **source*, **unsigned long** *sourceLen*);

unsigned long compressBound(unsigned long *sourceLen*);

DESCRIPTION

This manual page describes the *zlib* general purpose compression library, version 1.2.11. The *zlib* compression library provides in-memory compression and decompression functions, including integrity checks of the uncompressed data.

UTILITY FUNCTIONS

The **compress** function compresses the source buffer into the destination buffer. *sourceLen* is the byte length of the source buffer. Upon entry, *destLen* is the total size of the destination buffer, which must be at least the value returned by **compressBound(sourceLen)**. Upon exit, *destLen* is the actual size of the compressed data.

compress returns **Z_OK** if successful, **Z_MEM_ERROR** if there was not enough memory, or **Z_BUF_ERROR** if there was not enough room in the output buffer.

compressBound returns an upper bound on the compressed size after **compress** on *sourceLen* bytes. It would be used before a **compress** call to allocate the destination buffer.

NAME

memcpy – copy memory area

SYNOPSIS

#include <string.h>

void *memcpy(void **dest*, **const void** **src*, **size_t** *n*);

DESCRIPTION

The **memcpy**() function copies *n* bytes from memory area *src* to memory area *dest*. The memory areas must not overlap. Use **memmove**(3) if the memory areas do overlap.

RETURN VALUE

The **memcpy**() function returns a pointer to *dest*.

NOTES

Failure to observe the requirement that the memory areas do not overlap has been the source of significant bugs. (POSIX and the C standards are explicit that employing **memcpy**() with overlapping areas produces undefined behavior.) Most notably, in glibc 2.13 a performance optimization of **memcpy**() on some platforms (including x86-64) included changing the order in which bytes were copied from *src* to *dest*.

NAME

malloc, free, realloc – allocate and free dynamic memory

SYNOPSIS

#include <stdlib.h>

void *malloc(size_t *size*);
void free(void **ptr*);
void *realloc(void **ptr*, **size_t** *size*);

DESCRIPTION

The **malloc**() function allocates *size* bytes and returns a pointer to the allocated memory. *The memory is not initialized*. If *size* is 0, then **malloc**() returns either NULL, or a unique pointer value that can later be successfully passed to **free**().

The **free**() function frees the memory space pointed to by *ptr*, which must have been returned by a previous call to **malloc**(), or **realloc**(). Otherwise, or if *free(ptr)* has already been called before, undefined behavior occurs. If *ptr* is NULL, no operation is performed.

The **realloc**() function changes the size of the memory block pointed to by *ptr* to *size* bytes. The contents will be unchanged in the range from the start of the region up to the minimum of the old and new sizes. If the new size is larger than the old size, the added memory will *not* be initialized. If *ptr* is NULL, then the call is equivalent to *malloc(size)*, for all values of *size*; if *size* is equal to zero, and *ptr* is not NULL, then the call is equivalent to *free(ptr)*. Unless *ptr* is NULL, it must have been returned by an earlier call to **malloc**(), or **realloc**(). If the area pointed to was moved, a *free(ptr)* is done.

RETURN VALUE

The **malloc**() function return a pointer to the allocated memory, which is suitably aligned for any built-in type. On error, these functions return NULL. NULL may also be returned by a successful call to **malloc**() with a *size* of zero.

The **free**() function returns no value.

The **realloc**() function returns a pointer to the newly allocated memory, which is suitably aligned for any built-in type, or NULL if the request failed. The returned pointer may be the same as *ptr* if the allocation was not moved (e.g., there was room to expand the allocation in-place), or different from *ptr* if the allocation was moved to a new address. If *size* was equal to 0, either NULL or a pointer suitable to be passed to **free**() is returned. If **realloc**() fails, the original block is left untouched; it is not freed or moved.

ERRORS

malloc() and **realloc**() can fail with the following error:

ENOMEM
Out of memory. Possibly, the application hit the **RLIMIT_AS** or **RLIMIT_DATA** limit described in **getrlimit**(2).

NAME

pthread_create − create a new thread

SYNOPSIS

#include <pthread.h>

int pthread_create(pthread_t ***_thread_, const pthread_attr_t ***_attr_**,**
void *(*_start_routine_**) (void *), void ***_arg_**);**

DESCRIPTION

The **pthread_create**() function starts a new thread in the calling process. The new thread starts execution by invoking _start_routine_(); _arg_ is passed as the sole argument of _start_routine_().

The new thread terminates in one of the following ways:

* It calls **pthread_exit**(3), specifying an exit status value that is available to another thread in the same process that calls **pthread_join**(3).
* It returns from _start_routine_(). This is equivalent to calling **pthread_exit**(3) with the value supplied in the _return_ statement.
* It is canceled (see **pthread_cancel**(3)).
* Any of the threads in the process calls **exit**(3), or the main thread performs a return from _main_(). This causes the termination of all threads in the process.

The _attr_ argument points to a _pthread_attr_t_ structure whose contents are used at thread creation time to determine attributes for the new thread; this structure is initialized using **pthread_attr_init**(3) and related functions. If _attr_ is NULL, then the thread is created with default attributes.

Before returning, a successful call to **pthread_create**() stores the ID of the new thread in the buffer pointed to by _thread_; this identifier is used to refer to the thread in subsequent calls to other pthreads functions.

RETURN VALUE

On success, **pthread_create**() returns 0; on error, it returns an error number, and the contents of *_thread_ are undefined.

ERRORS

EAGAIN
Insufficient resources to create another thread.

EINVAL
Invalid settings in _attr_.

EPERM
No permission to set the scheduling policy and parameters specified in _attr_.

NOTES

A thread may either be _joinable_ or _detached_. If a thread is joinable, then another thread can call **pthread_join**(3) to wait for the thread to terminate and fetch its exit status. Only when a terminated joinable thread has been joined are the last of its resources released back to the system. When a detached thread terminates, its resources are automatically released back to the system: it is not possible to join with the thread in order to obtain its exit status. Making a thread detached is useful for some types of daemon threads whose exit status the application does not need to care about. By default, a new thread is created in a joinable state, unless _attr_ was set to create the thread in a detached state (using **pthread_attr_setdetachstate**(3)).

NAME

pthread_join − join with a terminated thread

SYNOPSIS

#include <pthread.h>

int pthread_join(pthread_t _thread_**, void ****_retval_**);**

Compile and link with _−pthread_.

DESCRIPTION

The **pthread_join**() function waits for the thread specified by _thread_ to terminate. If that thread has already terminated, then **pthread_join**() returns immediately. The thread specified by _thread_ must be joinable.

If _retval_ is not NULL, then **pthread_join**() copies the exit status of the target thread (i.e., the value that the target thread supplied to **pthread_exit**(3)) into the location pointed to by _retval_. If the target thread was canceled, then **PTHREAD_CANCELED** is placed in the location pointed to by _retval_.

If multiple threads simultaneously try to join with the same thread, the results are undefined. If the thread calling **pthread_join**() is canceled, then the target thread will remain joinable (i.e., it will not be detached).

RETURN VALUE

On success, **pthread_join**() returns 0; on error, it returns an error number.

ERRORS

EDEADLK
A deadlock was detected (e.g., two threads tried to join with each other); or _thread_ specifies the calling thread.

EINVAL
thread is not a joinable thread.

EINVAL
Another thread is already waiting to join with this thread.

ESRCH
No thread with the ID _thread_ could be found.

NOTES

After a successful call to **pthread_join**(), the caller is guaranteed that the target thread has terminated. The caller may then choose to do any clean-up that is required after termination of the thread (e.g., freeing memory or other resources that were allocated to the target thread).

Joining with a thread that has previously been joined results in undefined behavior.

Failure to join with a thread that is joinable (i.e., one that is not detached), produces a "zombie thread". Avoid doing this, since each zombie thread consumes some system resources, and when enough zombie threads have accumulated, it will no longer be possible to create new threads (or processes).

NAME

pthread_mutex_lock, pthread_mutex_trylock, pthread_mutex_unlock – lock and unlock a mutex

SYNOPSIS

#include <pthread.h>

int pthread_mutex_lock(pthread_mutex_t ****mutex*);**
int pthread_mutex_trylock(pthread_mutex_t ****mutex*);**
int pthread_mutex_unlock(pthread_mutex_t ****mutex*);**

DESCRIPTION

The mutex object referenced by *mutex* shall be locked by calling *pthread_mutex_lock*(). If the mutex is already locked, the calling thread shall block until the mutex becomes available. This operation shall return with the mutex object referenced by *mutex* in the locked state with the calling thread as its owner.

The *pthread_mutex_trylock*() function shall be equivalent to *pthread_mutex_lock*(), except that if the mutex object referenced by *mutex* is currently locked (by any thread, including the current thread), the call shall return immediately.

The *pthread_mutex_unlock*() function shall release the mutex object referenced by *mutex*. If there are threads blocked on the mutex object referenced by *mutex* when *pthread_mutex_unlock*() is called, resulting in the mutex becoming available, the scheduling policy shall determine which thread shall acquire the mutex.

If a signal is delivered to a thread waiting for a mutex, upon return from the signal handler the thread shall resume waiting for the mutex as if it was not interrupted.

RETURN VALUE

If successful, the *pthread_mutex_lock*() and *pthread_mutex_unlock*() functions shall return zero; otherwise, an error number shall be returned to indicate the error.

The *pthread_mutex_trylock*() function shall return zero if a lock on the mutex object referenced by *mutex* is acquired. Otherwise, an error number is returned to indicate the error.

COPYRIGHT

NAME

read – read from a file descriptor

SYNOPSIS

#include <unistd.h>

ssize_t read(int *fd*, **void ****buf*, **size_t** *count***);**

DESCRIPTION

read() attempts to read up to *count* bytes from file descriptor *fd* into the buffer starting at *buf*.

On files that support seeking, the read operation commences at the current file offset, and the file offset is incremented by the number of bytes read. If the current file offset is at or past the end of file, no bytes are read, and **read**() returns zero.

If *count* is zero, **read**() *may* detect the errors described below. In the absence of any errors, or if **read**() does not check for errors, a **read**() with a *count* of 0 returns zero and has no other effects.

RETURN VALUE

On success, the number of bytes read is returned (zero indicates end of file), and the file position is advanced by this number. It is not an error if this number is smaller than the number of bytes requested; this may happen for example because fewer bytes are actually available right now (maybe because we were close to end-of-file, or because we are reading from a pipe, or from a terminal), or because **read**() was interrupted by a signal. On error, −1 is returned, and *errno* is set appropriately. In this case, it is left unspecified whether the file position (if any) changes.

NAME

write – write to a file descriptor

SYNOPSIS

#include <unistd.h>

ssize_t write(int *fd*, **const void ****buf*, **size_t** *count***);**

DESCRIPTION

write() writes up to *count* bytes from the buffer pointed *buf* to the file referred to by the file descriptor *fd*.

The number of bytes written may be less than *count* if, for example, there is insufficient space on the underlying physical medium, or the **RLIMIT_FSIZE** resource limit is encountered (see **setrlimit**(2)), or the call was interrupted by a signal handler after having written less than *count* bytes.

For a seekable file (i.e., one to which **lseek**(2) may be applied, for example, a regular file) writing takes place at the current file offset, and the file offset is incremented by the number of bytes actually written. If the file was **open**(2)ed with **O_APPEND**, the file offset is first set to the end of the file before writing. The adjustment of the file offset and the write operation are performed as an atomic step.

RETURN VALUE

On success, the number of bytes written is returned (zero indicates nothing was written). On error, −1 is returned, and *errno* is set appropriately.

If *count* is zero and *fd* refers to a regular file, then **write**() may return a failure status if an error is detected. If no errors are detected, 0 will be returned without causing any other effect. If *count* is zero and *fd* refers to a file other than a regular file, the results are not specified.

NAME
stat, fstat – get file status

SYNOPSIS
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

int stat(const char ****pathname*, struct stat ****buf***);**
int fstat(int *fd***, struct stat ****buf***);**

DESCRIPTION
These functions return information about a file, in the buffer pointed to by *buf*. No permissions are required on the file itself, but—in the case of **stat**()—execute (search) permission is required on all of the directories in *pathname* that lead to the file.

stat() retrieves information about the file pointed to by *pathname*.

fstat() is identical to **stat**(), except that the file about which information is to be retrieved is specified by the file descriptor *fd*.

All of these system calls return a *stat* structure, which contains the following fields:

```
struct stat {
    dev_t     st_dev;         /* ID of device containing file */
    ino_t     st_ino;         /* inode number */
    mode_t    st_mode;        /* file type and mode */
    nlink_t   st_nlink;       /* number of hard links */
    uid_t     st_uid;         /* user ID of owner */
    gid_t     st_gid;         /* group ID of owner */
    dev_t     st_rdev;        /* device ID (if special file) */
    off_t     st_size;        /* total size, in bytes */
    blksize_t st_blksize;     /* blocksize for filesystem I/O */
    blkcnt_t  st_blocks;      /* number of 512B blocks allocated */

    time_t st_atime;  /* Time of last access */
    time_t st_mtime;  /* Time of last modification */
    time_t st_ctime;  /* Time of last status change */
};
```

The field *st_atime* is changed by file accesses, for example, by **execve**(2), **mknod**(2), **pipe**(2), **utime**(2) and **read**(2) (of more than zero bytes). Other routines, like **mmap**(2), may or may not update *st_atime*.

The field *st_mtime* is changed by file modifications, for example, by **mknod**(2), **truncate**(2), **utime**(2) and **write**(2) (of more than zero bytes). Moreover, *st_mtime* of a directory is changed by the creation or deletion of files in that directory. The *st_mtime* field is *not* changed for changes in owner, group, hard link count, or mode.

The field *st_ctime* is changed by writing or by setting inode information (i.e., owner, group, link count, mode, etc.).

RETURN VALUE
On success, zero is returned. On error, −1 is returned, and *errno* is set appropriately.

NAME
strcat, strncat – concatenate two strings

SYNOPSIS
#include <string.h>

char *strcat(char ****dest*, const char ****src***);**

char *strncat(char ****dest*, const char ****src***, size_t** *n***);**

DESCRIPTION
The **strcat**() function appends the *src* string to the *dest* string, overwriting the terminating null byte ('\0') at the end of *dest*, and then adds a terminating null byte. The strings may not overlap, and the *dest* string must have enough space for the result. If *dest* is not large enough, program behavior is unpredictable; *buffer overruns are a favorite avenue for attacking secure programs*.

The **strncat**() function is similar, except that

* it will use at most *n* bytes from *src*; and
* *src* does not need to be null-terminated if it contains *n* or more bytes.

As with **strcat**(), the resulting string in *dest* is always null-terminated.

If *src* contains *n* or more bytes, **strncat**() writes *n+1* bytes to *dest* (*n* from *src* plus the terminating null byte). Therefore, the size of *dest* must be at least *strlen(dest)+n+1*.

RETURN VALUE
The **strcat**() and **strncat**() functions return a pointer to the resulting string *dest*.

NAME
close – close a file descriptor

SYNOPSIS
#include <unistd.h>

int close(int *fd***);**

DESCRIPTION
close() closes a file descriptor, so that it no longer refers to any file and may be reused. Any record locks (see **fcntl**(2)) held on the file it was associated with, and owned by the process, are removed (regardless of the file descriptor that was used to obtain the lock).

If *fd* is the last file descriptor referring to the underlying open file description (see **open**(2)), the resources associated with the open file description are freed; if the file descriptor was the last reference to a file which has been removed using **unlink**(2), the file is deleted.

RETURN VALUE
close() returns zero on success. On error, −1 is returned, and *errno* is set appropriately.