FIXME

Technische Universität Braunschweig

Probeklausur

	erreichbare Punkte	erhaltene Punkte			
Aufgabe 1	12	**1**	**2**		
		6	6		
Aufgabe 2	22	**M**	**S**	**W**	
		10.5	6.5	5	
Aufgabe 3	3	**1**			
		3			
Aufgabe 4	8	**1**			
		8			
Summe	45				

(Name) (Vorname) (Matrikel-Nr.)

(HBK Matrikel-Nr.) (Studiengang) (Semester)

Durch meine Unterschrift bestätige ich

- den Empfang der vollständigen Klausur (12 Seiten inklusive Deckblatt),

Braunschweig, 25.01.2025 .....

(Unterschrift)

Aufgabe 1: Ankreuzfragen (12 Punkte)

(A1.1) Einfachauswahlfragen (6 Punkte)

Bei den Einfachauswahlfragen in dieser Aufgabe ist jeweils nur **eine** richtige Antwort eindeutig anzukreuzen. Auf die richtige Antwort gibt es die angegebene Punktzahl.

Wollen Sie eine Antwort korrigieren, streichen Sie bitte die falsche Antwort mit drei waagerechten Strichen durch (⊗) und kreuzen die richtige an.

Lesen Sie die Frage genau, bevor Sie antworten.

a) Zu welchem Zweck werden die Semaphoren `slots` und `elem` hier verwendet?

2 Punkte- 2

```
semaphore slots = N;
semaphore elem  = 0;

void insert( char item ){
  down(&slots);
  Buffer[++in % N] = item;
  up(&elem);
}
```

```
char remove( void ){
  char item;
  down(&elem);
  item = Buffer[++out % N]
  up(&slots);
  return item;
}
```

- ○ Koordinierter Zugriff auf wiederverwendbare Betriebsmittel
- ⊗ Logische Abhängigkeit von konsumierbare Betriebsmittel
- ○ Lost-Update Vermeidung
- ○ Atomare Ausführung

b) Ein Prozess wird vom Zustand `blockiert` in den Zustand `bereit` überführt. Welche Aussage passt zu diesem Vorgang?

2 Punkte- 2

- ○ Es ist kein direkter Übergang von blockiert nach bereit möglich.
- ○ Ein anderer Prozess wurde vom Betriebssystem verdrängt und der erstgenannte Prozess wird nun auf der CPU eingelastet.
- ⊗ Der Prozess hat auf Daten von der Festplatte gewartet, die nun verfügbar sind.
- ○ Der Prozess wird wegen eines ungültigen Speicherzugriffs (Segmentation Fault) beendet.

c) Welche Aussagen zu hierarchischen Dateisystemen ist richtig?

2 Punkte- 2

- ○ Der Nachteil von hierarchischen Namensräumen besteht darin, dass das Dateisystem spezielle Funktionen zum Auflösen von Namenskonflikten implementieren muss
- ⊗ Hierarchische Dateisysteme ermöglichen es, dass gleiche Namen in verschiedenen Kontexten vorkommen dürfen.
- ○ Hierarchische Dateisysteme sind aufgrund ihrer flachen Namensräume besonders einfach zu implementieren und werden daher insbesondere für Mehrbenutzersysteme verwendet.
- ○ Symbolische Verweise auf Dateien übergeordneter Ebenen sind nicht zulässig, da ansonsten Zyklen entstehen könnten.

(A1.2) Mehrfachauswahlfragen (6 Punkte)

In dieser Aufgabe sind jeweils m Aussagen angegeben. Davon sind n ($0 \leq n \leq m$) Aussagen richtig. Kreuzen Sie jeweils an, ob die entsprechende Aussage richtig oder falsch ist.
Jede korrekte Antwort gibt 0,5 Punkte, jede falsche Antwort 0,5 Punkte Abzug. Nicht beantwortete Aussagen gehen neutral in die Bewertung ein. Eine Teilaufgabe wird minimal mit 0 Punkten gewertet, falsche Antworten wirken sich nicht auf andere Teilaufgaben aus.

Wollen Sie eine falsch angekreuzte Antwort korrigieren, streichen Sie bitte das Kreuz mit drei waagrechten Strichen durch (☒).

Lesen Sie die Frage genau, bevor Sie antworten.

a) Welche der genannten Aufgaben gehören zwingend zum Betriebssystemkern?

2 Punkte- 2

Richtig	Falsch	
☐	☒	Gerätetreiber
☒	☐	Multiplexing der Hardware
☐	☒	Benutzeroberfläche (z.Ḃ. Shell)
☒	☐	Isolation zwischen Prozessen

b) Welche der genannten Aussagen zum Thema Semaphore sind richtig?

2 Punkte- 2

Richtig	Falsch	
☒	☐	Ein Semaphor kann verwendet werden, um gegenseitigen Ausschluss zu implementieren.
☐	☒	Die `down()` Operation kann nur von dem Kontrollfluss aufgerufen werden, der sich gerade im kritischen Abschnitt befindet.
☐	☒	Die `up()` Funktion eines Semaphors erhöht den Zähler der Semaphore falls möglich und wartet sonst bis dies möglich ist.
☒	☐	Ein Semaphor kann zum Signalisieren von Ereignissen verwendet werden.

c) Beurteilen Sie folgende Aussagen zu UNIX-Prozessen

2 Punkte- 2

Richtig	Falsch	
☐	☒	Das Betriebssystem überwacht laufend im Hintergrund, dass kein Prozess auf Ressourcen eines Anderen zugreift.
☒	☐	Ein Prozess ist ein virtueller Computer.
☐	☒	Das Betriebssystem übersetzt den Programmcode des Programms.
☒	☐	Das Betriebssystem erweitert den Befehlssatz des realen Prozessors.

Aufgabe 2: Programmieraufgabe – `linkr` (22 Punkte)

Aufgabe 2.1: Dateisysteminteraktion

Schreiben Sie ein Programm `linkr`, welches gegebene Pfade und Unterverzeichnisse auf symbolische Links untersucht.

Die Funktion `void linkr(char *path)` iteriert über die Einträge eines Verzeichnisses.

- Für jeden Datei vom Typ Symlink soll mittels `print_link(char *path)` der Pfad der Zieldatei ausgelesen und anschließend ausgegeben werden.
- Für jede Datei vom Typ Verzeichnis soll die Funktion `process_subdir(char *path)` aufgerufen werden.
- Andere Dateitypen sollen ignoriert werden.

Das Auslesen und Ausgeben eines Links soll in der Funktion `void print_link(char *path)` implementiert werden. Diese liest den Zielpfad aus dem Link aus. Pfade können maximal 4095 Zeichen lang sein. War das Lesen erfolgreich, so sollen Dateipfad und Zielpfad ausgegeben werden. Im Fehlerfall soll das Programm mit Fehlerausgabe beendet werden.

Die Funktion `void process_subdir(char *path)` kann als gegeben angesehen werden.

Beispiel:

```
nutzer@x1:~ $ ./linkr dir1 dir2
dir1/lnk1 -> /foo/bar
dir2/test -> ../tmp
```

Beachten Sie die beigefügten Man-Pages (am Ende der Klausur). Diese geben Hinweise auf die Verwendung von Bibliotheksfunktionen und deren Verhalten. Achten Sie weiterhin darauf, dass Sie eventuelle Fehlersituationen der Bibliotheksfunktionen im Sinne der Aufgabenstellung behandeln und das Programm mit Fehlerausgabe terminieren.

Verwenden Sie zum Bearbeiten dieser Aufgabe die Vorgabe auf der Seite 6. Ergänzen Sie jeweils die fehlenden Teile des Programmtextes, sodass ein Programmablauf im Sinne der Aufgabenstellung entsteht.

Vorgabe zu 2.1: Dateisysteminteraktion

10,5 Punkte-
10.5

```
void wait_for_subdirs(void);
void print_link(char *path);
void process_subdir(char *path);
void die(char *cause);
int readlink(char *path, char *buf, int bufsize);

void linkr(char *path) {
    fprintf(stderr, "linkr %10d %s\n", getpid(), path);
    DIR *dir = opendir(path)                    ✓;
    if ( dir == NULL                    ✓ ) {
        die(path);
    }
    struct dirent *ent;
    while( (ent = readdir(dir)) != NULL                    ✓✓ ) {
        char file_path[4096]                    ✓; //file_path memory
        if (ent->d_name[0] == '.')
            continue;
        strcpy(file_path, path);  //file_path construction
        strcat(file_path, "/");
        strcat(file_path, ent->d_name);
        if ( ent->d_type == DT_LNK                    ✓ ) {
            print_link(file_path);
        }
        if ( ent->d_type == DT_DIR                    ✓ ) {
            process_subdir(file_path);
        }
    }
    closedir(dir)                    ✓;
}

void print_link(char *path) {
    char dest[4096]                    ✓; // memory for target path
    int n = readlink(path, dest, sizeof(dest)        )                ✓✓;
    if ( n != -1                ✓ ) {
        printf("%s -> %s\n", path, dest                ✓ );
    }
}
```

Aufgabe 2.2: Prozesse

Schreiben Sie die fehlenden Funktionen `process_subdir(char *path)` und `wait_for_subdirs()` für das `linkr`-Programm.

Die Funktion `process_subdir(char *path)` soll für das übergebene Verzeichnis einen neuen Prozess starten, der die `linkr`-Funktion für dieses ausführt. Der Elternprozess merkt sich jeweils die ID des Kindes und das zu bearbeitende Verzeichnis mittels der gegebenen Funktion `add_job(pid_t child, char *path)`. Nachdem das Verzeichnis abgearbeitet ist, wartet der Prozess mittels `wait_for_subdirs()` darauf, dass die Kindprozesse beendet werden.

Die Funktion `wait_for_subdirs()` soll auf alle Kindprozesse warten. Zu jedem beendeten Prozess soll eine Zeile mit dessen PID und Pfad ausgegeben werden. Die Pfade sind mittels `char * get_job(pid)` abrufbar. Der Speicher mit dem Pfad muss freigegeben werden. Hat `errno` nach dem Warten auf ein Kind den Wert `ECHILD`, so sind keine weiteren Kinder mehr ausstehend.

Beachten Sie die beigefügten Man-Pages (am Ende der Klausur). Diese geben Hinweise auf die Verwendung von Bibliotheksfunktionen und deren Verhalten. Achten Sie weiterhin darauf, dass Sie eventuelle Fehlersituationen der Bibliotheksfunktionen im Sinne der Aufgabenstellung behandeln und das Programm mit Fehlermeldung beenden.

Verwenden Sie zum Bearbeiten dieser Aufgabe die Vorgabe auf der Seite 8. Ergänzen Sie jeweils die fehlenden Teile des Programmtextes, sodass ein Programmablauf im Sinne der Aufgabenstellung entsteht.

Vorgabe zu 2.2: Prozesse

```
void die(char *cause);  //Exit with message
void linkr(char *path);
void add_job(int pid, char *path);
char * get_job(pid_t pid);
```

(A2.2) Unterverzeichnis abarbeiten (6,5 Punkte)

```
void process_subdir(char *path) {
    // 1P: syscall
    pid_t child = fork();
    // 1P: 3 Fälle
    if (child < 0) {
        // 0.5P Fehler
        die("fork");
    }
    if (child > 0) {
        // 1P: enqueue
        add_job(child, path);
        return;
    } else {
        // 1P:
        linkr(path);
        // 1P:
        wait_for_subdirs();
        // 1P: Kind darf nicht weiter durch die Hierarchie
        // nach oben
        exit(EXIT_SUCCESS);
    }
}
```

S: 6.5

(A2.3) Warten auf Unterverzeichnisse (5 Punkte)

```
void wait_for_subdirs() {
    pid_t child;
    int status;
    // 1P: Schleife, Bedingung
    // 1P: wait, param, ret
    while ((child = wait(&status)) != -1) {
        // 1P:
        char *path = get_job(child);
        // 1P:
        printf("%d finished %s \n", child, path);
        // 1P:
        free(path);
    }
    if (errno == ECHILD)
        return;
    die("waitpid");
}
```

W: 5

Aufgabe 3: Nebenläufigkeit (3 Punkte)

Welches Problem kann hier auftreten? Wo tritt das Problem auf? Wie kann man dies vermeiden? Ihre Antworten sollen kurz und prägnant sein (Stichworte, kurze Sätze).

3 Punkte-
3

```
unsigned char pos = 0; int results[256];

                //thread 1                          //thread 2
                do {                              do {
                  waitForJob();                     waitForJob();
                  int r = doJob();                  int r = doJob();
                  results[++pos] = r;               results[++pos] = r;
                } while(1);                       } while(1);
```

Problem: Lost update

Ort: ++pos

Lösung: Gegenseitiger Ausschluss aus dem kritischen Gebiet

Also atomares erhöhen des Zählers, z.B. mit Sem/Mutex

Aufgabe 4: Dateisysteme (8 Punkte)

8 Punkte-
8

Gegeben ist die folgende Ausgabe des Kommandos `ls -aoRi /tmp/folder` (rekursiv absteigende Ausgabe aller Dateien und Verzeichnisse unter `/tmp/folder` mit Angabe der Inode-Nummer, des Referenzzählers und der Dateigröße) auf einem Linux-System.

```
/tmp/folder:
total 80
  41 drwxr-xr-x  3 chrdietr  4096 Mär  4 14:55 .
  90 drwxrwxrwt 36 root    57344 Mär  4 15:56 ..
  42 drwxr-xr-x  2 chrdietr  4096 Mär  4 14:53 foo
  71 -rw-r--r--  1 chrdietr 10678 Mär  2 18:21 listings.c
  79 lrwxrwxrwx  2 chrdietr    10 Mär  2 22:21 rebos -> /bs/repos

/tmp/folder/foo:
total 44
  42 drwxr-xr-x 2 chrdietr  4096 Mär  4 14:53 .
  41 drwxr-xr-x 3 chrdietr  4096 Mär  4 14:55 ..
  75 -rw-r--r-- 1 chrdietr 34005 Mär  2 18:22 dump.hex
  79 lrwxrwxrwx 2 chrdietr    10 Mär  2 22:21 rebos -> /bs/repos
```

Ergänzen Sie im weißen Bereich die auf der Vorlage im grauen Bereich bereits angefangene Skizze der Inodes und Datenblöcke des Linux-Dateisystems um alle entsprechenden Informationen, die aus obiger Ausgabe entnommen werden können.