UNIVERSITEIT TWENTE.

Examination Operating Systems

Bachelor year 2, Computer Science, EWI

Module/course code: Computer Systems
Date: 11 November 2016
Time: 13:45-15:45 (+25% for students who may use extra time)
Module-coördinator: Andre Kokkeler
Instructor: Pieter Hartel
100% of the marks: 60 credits

Type of test:

- Closed book

Allowed aids during the test:

- Nothing

Attachments:

- Various Linux Manual pages

Additional remarks:

- Read these instructions and the questions carefully! If the questions are unclear, you can ask for clarification.
- Please make sure that your name and student number appear on all answer sheets.
- Try to give precise answers using appropriate terminology. For multiple-choice questions there may be more than one correct answer; all of these must be selected for full marks.
- Unreadable or extremely long answers will not be marked. Multiple-choice answers that are ambiguous will not be marked either.
- You are only allowed to use your writing materials and the Linux man pages provided during the exam.
- Feel free to give your answers in English.

Nr:	1.3

Q:	Consider the two C-programs Uname.c and Vname.c below:
	
	`/* Uname.c */`
	`#include <stdio.h>`
	`#include <sys/utsname.h>`
	
	`int main(int argc, char * argv[]) {`
	`  struct utsname u;`
	`  if(uname(&u) == 0) {`
	`  printf("%s %s %s %s\n",`
	`  u.nodename, u.sysname,`
	`  u.release, u.machine);`
	`  }`
	`  return 0;`
	`}`
	
	`/* Vname.c */`
	`#include <stdio.h>`
	`#include <stdlib.h>`
	`#include <sys/utsname.h>`
	
	`int main(int argc, char * argv[]) {`
	`  struct utsname *v = malloc(sizeof(struct utsname));`
	`  if(uname(v) == 0) {`
	`  printf("%s %s %s %s\n",`
	`  v->nodename, v->sysname,`
	`  v->release, v->machine);`
	`  }`
	`  return 0;`
	`}`
	
	(a) What is the main difference between the two programs? Explain.
	(b) What is the main similarity of the two programs? Explain.

C:	7 credits

Nr:	3.9

Q:	Consider the C program fragment below:
	
	`int main(int argc, char *argv[]) {`
	`  pid_t pid=fork();`
	`  printf("%s\n", argv[0]);`
	`  if (pid==0) {`
	`  static char *argv[]={"echo","Foo",NULL};`
	`  execv("/bin/echo",argv);`
	`  exit(127);`
	`  } else {`
	`  waitpid(pid,0,0);`
	`  }`
	`  return 0;`
	`}`
	
	Assume that the compiled version of the program is executed as "`./a.out A B`".
	(a) What is the purpose of the first argument to the execv system call?
	(b) What will be the contents of the argv array to the main function of the echo program? Explain.
	(c) What is the purpose of the call to waitpid?
	(d) Under which circumstances might the exit function be called?

C:	7 credits

Nr:	4.10

Q:	Consider the C program fragment below:
	
	`void *tproc(void *arg) {`
	`  static char *argv[]={"echo","Foo",NULL};`
	`  execv("/bin/echo",argv);`
	`}`
	
	`int main(int argc, char *argv[]) {`
	`  int targ = 0;`
	`  pthread_t tid;`
	`  pthread_create(&tid, NULL, &tproc, &targ);`
	`  printf("%s\n", argv[0]);`
	`  pthread_join(tid,NULL);`
	`  return 0;`
	`}`
	
	(a) How many threads will be created when this program is run by the shell? Explain.
	(b) What is the output of the program? Explain.
	(c) Is the output always the same? Explain.

C:	7 credits

Nr:	5.10
Q:	Assume that the C-program of which the main fragment is shown below runs on a single core computer.

```
#define P 2
#define Q 4
#define R 7
#define M 1690

/* Burn about N * 10 ms CPU time */
void loop(int N) {
    int i, j, k ;
    for(i = 0; i < N; i++) {
        for(j = 0; j < M; j++) {
            for(k = 0; k < M; k++) {
            }
        }
    }
}

int main(int argc, char *argv[]) {
    for( int p = 0; p < P; p++ ) {
        for( int q = 0; q < Q; q++ ) {
            int child = fork();
            if (child == 0) {
                child = getpid();
                setpriority(PRIO_PROCESS, child, p) ;
                for(int r = 0; r < R; r++) {
                    loop( 100 );
                }
                exit(0) ;
            }
        }
    }
    return 0;
}
```

(a) How many child processes are created by the main process? Explain.
(b) Which of the child processes will terminate first and which will terminate last? Explain.
(c) Would your answer for (b) remain the same if the program would be run on an 8 core system with plenty of RAM? Explain.

C:	7 credits						

Nr:	6.8
Q:	A semaphore satisfies the following invariants: $S \geq 0$ $S = S_0 + \#Signals - \#Waits$ where S_0 is the initial value of S #Signals is the number of executed Signal(S) operations #Waits is the number of completed Wait(S) operations Given the two concurrent processes below, prove the mutual exclusion property, using the two semaphore invariants. S_0 is initialised to 1 before the processes start.

```
while(true) {
 a1: Non_Critical_Section_1;
 b1: Wait(S);
 c1: Critical_Section_1;
 d1: Signal(S);
}
```

```
while(true) {
 a2: Non_Critical_Section_2;
 b2: Wait(S);
 c2: Critical_Section_2;
 d2: Signal(S);
}
```

C:	7 credits

Nr:	7.6
Q:	Consider the program fragment below, representing a semaphore solution to the dining Philosophers problem. Assume that all semaphores have been initialised correctly and that there are three threads, one for each of the three philosophers with k=0, k=1 and k=2.

```
#define N 5
#define P 3

sem_t Room; /* Initialised to P-1 */
sem_t Fork[P]; /* initialized to 1 */

void *tphilosopher(void *ptr) {
    int i, k = *((int *) ptr);
    for(i = 1; i <= N; i++) {
        printf("Tnk %d %d\n", k, i);
        sem_wait(&Room) ;
        sem_wait(&Fork[k]) ;
        sem_wait(&Fork[(k+1) % P]) ;
        printf("Eat %d %d\n", k, i);
        sem_post(&Fork[k]) ;
        sem_post(&Fork[(k+1) % P]) ;
        sem_post(&Room) ;
    }
    pthread_exit(0);
}
```

(a) Give an example of the output of the program.
(b) Assume that N= infinity, then show that it possible for two of the philosophers to conspire so that a third philosopher will starve. Explain the scenario.
(c) How could the starvation problem be solved?

C:	7 credits

Nr:	9.7

Q: Consider the C-program fragment with numbered lines below:

```
 1  int main(int argc, char *argv[]) {
 2      int in = open(argv[1], O_RDONLY);
 3      int out = open(argv[2],
 4          O_RDWR|O_CREAT|O_TRUNC, 0666);
 5      size_t sz = lseek(in, 0, SEEK_END);
 6      lseek(out, sz - 1, SEEK_SET);
 7      write(out, "\0", 1);
 8      void *src = mmap(NULL, sz,
 9          PROT_READ, MAP_PRIVATE, in, 0);
10      void *tgt = mmap(NULL, sz,
11          PROT_WRITE, MAP_SHARED, out, 0);
12      memcpy(tgt, src, sz);
13      munmap(src, sz);
14      munmap(tgt, sz);
15      close(in);
16      close(out);
17      return 0;
18  }
```

The system call trace of the program obtained from "strace ./a.out Mmap.c Foo" lists the following system calls:

```
open("Mmap.c", O_RDONLY)                      = 3
open("Foo", O_RDWR|O_CREAT|O_TRUNC, 0666)     = 4
lseek(3, 0, SEEK_END)                         = 1393
lseek(4, 1392, SEEK_SET)                      = 1392
write(4, "\0", 1)                             = 1
mmap(NULL, 1393, PROT_READ, MAP_PRIVATE, 3, 0) =
0x7f1e10da0000
mmap(NULL, 1393, PROT_WRITE, MAP_SHARED, 4, 0) =
0x7f1e10d9f000
munmap(0x7f1e10da0000, 1393)                  = 0
munmap(0x7f1e10d9f000, 1393)                  = 0
close(3)                                      = 0
close(4)                                      = 0
```

(a) How many bytes long is the file Mmap.c?
(b) Why is there no read system call reading the Mmap.c file?
(c) Why is there no write system call writing out the file Foo?
(d) What is the purpose of the memcpy function?
(e) Why does the memcpy function not show up in the strace?

C:	7 credits						

Nr:	11.10
Q:	Consider the C-program fragment below:

```
int main(int argc, char * argv[]) {
    DIR *dirp = opendir(argv[1]) ;
    if ( dirp != NULL ) {
        struct dirent *dp ;
        while (dp = readdir(dirp)) {
            char t;
            switch( dp->d_type ) {
                case DT_BLK     : t = 'b' ; break ;
                case DT_CHR     : t = 'c' ; break ;
                case DT_DIR     : t = 'd' ; break ;
                case DT_FIFO    : t = 'p' ; break ;
                case DT_LNK     : t = 'l' ; break ;
                case DT_REG     : t = '-' ; break ;
                case DT_SOCK    : t = 's' ; break ;
                case DT_UNKNOWN : t = 'u' ; break ;
                default         : t = '?' ;
            }
            printf("%8d %c %s\n",
                (int)dp->d_ino, t, dp->d_name);
        }
        closedir(dirp);
    }
    return 0;
}
```

(a) When does the while loop terminate? Explain.
(b) What type of file would be labelled with a 'b'?
(c) What type of file would be labelled with a 'c'?
(d) What is printed by dp->d_ino?
(e) If the output contains the two lines below, which directory has been given as the first argument to the program?

```
2 d .
2 d ..
```

C:	7 credits

Nr:	12.6
Q:	The disk on a particular system has a **maximum** seek time of 11.68 ms, a data transfer time of 100MBytes per second and the disk spins at 7200 rpm. (a) What is the average latency of a disk read, assuming that all delays take place sequentially? (b) What is the average transfer rate if the average file is 1MByte?
C:	4 credits