UNIVERSITEIT TWENTE.

Examination Operating Systems

28 January 2014, Sport Centrum

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.

Your working time begins at 13:45 and ends at 17:15.

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 during the exam.

All answers must be given in English.

Nr: 1.3

Q: Consider the two C-programs Uname.c and Vname.c below:

```c
/* 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

a) het verschil is dat bij de eerste er het utsname struct is en in de 2e een pointer naar een utsname struct, hierdoor kun je gewoon u. doen, bij de 2e moet je v-> doen.

b) beide printen nodename, sysname, release en machine

Nr: 2.6

Q: Given the following C program fragment:

```
extern char etext, edata, end;
int a = 0xaaaa, b;

int main(int argc, char * argv[]) {
     int c = 0xcccc;
     int *d_ptr = (int*) malloc(sizeof(int));
     int *e_ptr = (int*) alloca(sizeof(int));
     b = 0xbbbb;
     *d_ptr = 0xdddd;
     *e_ptr = 0xeeee;
     printf("%p:a=%0x\n", &a, a);
     printf("%p:edata\n\n", &edata);
     printf("%p:b=%0x\n", &b, b);
     printf("%p:end\n\n", &end);
     printf("%p:d=%0x\n", d_ptr, *d_ptr);
     printf("%p:brk\n\n", sbrk(0));
     printf("%p:e=%0x\n", e_ptr, *e_ptr);
     printf("%p:argc=%0x\n", &argc, argc);
     printf("%p:c=%0x\n\n", &c, c);
     printf("%p:main\n", &main);
     printf("%p:etext\n\n", &etext);
     return 0;
}
```

(a) Give an example of the output of the program. Here the actual addresses are not important (you may even choose to use 1, 2, 3 etc), but the order of the addresses must be correct.

(b) Annotate each segment of the program's address space with its conventional Unix/Linux name.

(c) If you would run the program several times on a current Linux system with the latest protection, then which addresses would change and which would remain the same? Why?

text.
main
c
data
a
~~ex~~ bss
b
brk
heap
e
end
stack
d
argv
argc

C: 7 credits

a)

a=
edata
b=
end
d=
brk
e=
argc
c=

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`". (Hint: the NULL pointer at the end of the argv array indicates the end of the list of arguments).

(a) What is the purpose of the first argument to the execv system call?
(b) What will be the contents of the argv argument to the main function of the echo program?
(c) What is the purpose of the call to waitpid?

C:	7 credits							

Nr:	4.8
Q:	Consider the C and Java program fragments below:

```
#define N 5000
void* tproc(void *arg) {
    printf("Thread %d\n", *((int *)arg));
    return NULL;
}
int main(int argc, char * argv[]) {
    int i;
    int targ[N];
    pthread_t tid[N];
    for(i = 0; i < N; i++) {
        targ[i] = i*i;
        pthread_create(&(tid[i]), NULL, &tproc, &targ[i]);
    }
    for(i = 0; i < N; i++) {
        pthread_join(tid[i], NULL) != 0;
    }
    return 0;
}

class MyThread extends Thread {
    static final int N = 5000 ;
    int arg;
    public MyThread(int arg) {
        this.arg = arg;
    }
    public void run() {
       System.out.println("Thread " + arg);
    }
    public static void main(String [] args) {
        MyThread[] tid = new MyThread [N] ;
        for(int i = 0; i < N; i++) {
            tid[i] = new MyThread(i*i);
            tid[i].start();
        }
        for(int i = N-1; i >= 0; i--) {
            try { tid[i].join(); }
            catch(InterruptedException e) { }
        }
    }
}
```

(a) What is the main difference between the outputs produced by the two programs? Why?

(b) Which of the two programs is faster? Why?

C:	7 credits						

Nr:	4.9

Q: Consider the C program fragment below:

```
#define N 5
void* tproc(void *arg) {
    printf("Thread %d\n", *((int *)arg));
    return NULL;
}
int main(int argc, char * argv[]) {
    int i;
    int targ[N];
    pthread_t tid[N];
    for(i = 0; i < N; i++) {
        targ[i] = i*i;
        pthread_create(&(tid[i]), NULL, &tproc, &targ[i]);
    }
    for(i = 0; i < N; i++) {
        pthread_join(tid[i], NULL);
    }
    return 0;
}
```

(a) How many threads will be created when this program is run? Why?
(b) What is the output of the program? Why?
(c) Is the output always the same? Why?
(d) Explain why the argument of the `printf` statement is not simply: `* arg`.

C:	4 credits						

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

```
#define P 3
#define Q 3
#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 always remain the same if the program would be run on an 8 core system? 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.

```
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:	4 credits						

Nr:	6.12
Q:	Consider the Linux command pipeline below:

```
cat foo | sort | uniq -c | sort -rn | pr -2
```

Assume the file foo contains 7 lines as follows:

```
A
Bb
Ccc
Dddd
Ccc
Bb
A
```

(a) Show the intermediary results passing through each of the four pipes
(b) What is the final output of the pipeline?

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 1 */
sem_t Fork[P]; /* initialized to P-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) 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:	9.6
Q:	Consider the C-program fragment below:

```
#define N 5
#define P 4096

int main(int argc, char* argv[]) {
    int i, k;
    char buffer[N*P];
    struct rusage usage;
    getrusage(RUSAGE_SELF, &usage);
    k = usage.ru_minflt; /* number of page faults now */
    for (i=0; i < N*P; i++) {
        buffer[i] = 0;
        getrusage(RUSAGE_SELF, &usage);
        if( k != usage.ru_minflt ) {
            int f = usage.ru_minflt ;
            /* A: print i and f; B: save i and f */
            k = f;
        }
    }
    /* B: print the table with all i and f */
    return 0;
}
```

The programmer made two versions, A and B of the program. Instead of the comment /* save ... */ version A prints the value of the variables i and f, and version B saves both values in a small array for printing later, just before the return statement.

Output of version A:

```
i= a30, f=156
i= a31, f=161
i=1a30, f=162
i=2a30, f=163
i=3a30, f=164
```

Output of version B:

```
i=   0, f=153
i= 470, f=154
i=1470, f=155
i=2470, f=156
i=3470, f=157
```

(a) What is the main difference between the two programs? What is the cause of the difference?
(b) What is the main similarity of the two programs? Explain.

C:	7 credits						

Nr:	11.9
Q:	Consider the C program fragment below:

```
int main(int argc, char **argv) {
    int fd[2];
    pipe(fd);
    printf("top  %d\n", getpid());
    pid_t pid=fork();
    if(pid== 0) {
        close(fd[1]);
        read(fd[0], &pid, sizeof (int));
        printf("child  %d\n", pid);
        close(fd[0]);
    } else {
        close(fd[0]);
        printf("parent %d\n", pid);
        pid = getpid();
        write(fd[1], &pid, sizeof (int));
        close(fd[1]);
        waitpid(pid,0,0);
    }
    return 0;
}
```

(a) Which process prints the pid of the child? Why?
(b) Which process prints the pid of the parent? Why?

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: 12.7

Q: Consider the C-program fragment below when executed on a 32-bit machine:

```
#define M 1024*256
#define N ???? /* number of buffers written */

int main(int argc, char *argv[]) {
    int out = open(argv[1], O_RDWR|O_CREAT|O_TRUNC, 0666);
    int i,k;
    int buf[M]; /* One MByte */
    for(i=0;i<N;i++) {
        for(k=0;k<M;k++) {
            buf[k] = i*N+k;
        }
        write(out,buf,sizeof(buf));
    }
    if(argc>=3) {
        fdatasync(out);
        posix_fadvise(out, 0,0,POSIX_FADV_DONTNEED);
    }
    close(out);
    return 0;
}
```

(a) What is the purpose of the fdatasync function?

(b) What is the purpose of the posix_fadvise function?

(c) Executing the script in the left column below shows that the amount of cached disk space at each occurrence of the free command is as indicated in the right column. What is the most likely value for N? Explain.

Command	Cached (MB)
free -m	9449
./a.out Foo	
free -m	9849
cp Foo Foo1	
free -m	10249
./a.out Bar x	
free -m	10249
cp Bar Bar1	
free -m	11049

400 400 0 800

C: 7 credits

Nr: 14.6

Q:

(a) Identity verification can be done with three modalities, i.e. something you have, something you are, and something you know. Give two examples of each modality.
(b) What is the main advantage of an identity verification system that uses two modalities instead of just one?
(c) And what is the main disadvantage?

C: 3 credits

Nr: 15.8

Q: Consider the C program fragment below:

```c
void foo(const char *fr) {
    char to[2];
    strcpy(to, fr);
}

int main(int argc, char * argv[]) {
    char fr[] = "abcdefghijklmnopqrstuvwxyz";
    char to[2] ;
    strcpy(to,fr) ;
    printf("to=%p=%s\nfr=%p=%s\n",
        (void*)to, to, (void*)fr, fr);
    fflush(stdout);
    foo(to);
    return 0;
}
```

(a) What is the output of the program? Why?
(b) For what purpose are more sophisticated versions of this type of program used?

C: 7 credits

Nr: LAB2014.1

Q: Answer the following questions about VTreeFS:

(a) What are the main features of the VTreeFS library?
(b) What is an inode?
(c) What is the "inode number" used for?
(d) What happens if the VTreeFS is running out of inodes?

C: 6 credits