UNIVERSITEIT TWENTE.

Ten sample questions and answers from the Examination Operating Systems of 29 January 2013

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:	1.1						
U:	1/2013, Model						

Q:	Consider the following code
	`for(i=0; i<20; i++)`
	`    for(j=0; j<10; j++)`
	`        a[i] = a[i] * j;`
	(a) Give an example of the spatial locality in the code
	(b) Given an example of the temporal locality in the code						

C:	2 credits	T:	T	D:	5 min	L:	Easy
A:	(a) An example of spatial locality: the array elements are read and written in sequence determined by the **outer loop**. (1 credit) (b) An example of temporal locality: the **inner loop** accesses i & j and the same array element 10 (20) times in a row, or the variables i and j (1 credit)						

Nr:	3.7
U:	1/2013, Model

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;
}
```

Suppose the compiled version of the program is executed as "`./a.out Foo Bar`".

(a) What is the actual output of the program? Why?
(b) Is the exit function ever called? Why?

C:	7 credits	T:	T	D:	5 min	L:	Easy

A:

(a) The actual output is

```
        ./a.out    ← forgetting one of these costs 1 credit
        ./a.out
        Foo
```

Both the parent and the child process print argv[0], which will be ./a.out. Then the child process will execute the echo command that prints the argument Foo. (4 credits)

(b) exit(127) is never called because **exec loads a new image** in the current process, unless of course the fork() or the execv() fails…. (3 credits)

Nr:	4.3						
U:	1/2013, Model						
Q:	Provide two examples of an application where multithreading does not provide better performance than a single-threaded solution. Explain why.						
C:	2 credits	T:	T	D:	4 min	L:	Medium
A:	1. Any kind of sequential program is not a good candidate to be threaded. An example of this is a program that calculates an individual tax return (1 credit) 2. Another example is a "shell" program such as the C-shell or Korn shell. Such a program must closely monitor its own working space such as open files, environment variables, and current working directory (1 credit)						

Nr:	5.1
U:	1/2013, Model

Q:	Discuss briefly how the following pairs of scheduling criteria conflict in certain settings: (a) CPU utilization and response time (b) Average turnaround time and maximum waiting time (c) I/O device utilization and CPU utilization						
C:	3 credits	T:	T	D:	6 min	L:	Medium
A:	(a) CPU utilization and response time: CPU utilization is **increased** if the **overhead** associated with **context switching** is **minimized**. The context switching overheads could be lowered by performing **context switches infrequently**. This could, however, result in **increasing** the **response time** for processes. (1 credit) (b) Average turnaround time and maximum waiting time: Average turnaround time is minimized by executing the **shortest tasks first**. Such a scheduling policy could, however, **starve long-running tasks** and thereby **increase** their **waiting time**. (1 credit) (c) I/O device utilization and CPU utilization: CPU utilization is maximized by running **long-running CPU-bound tasks without performing context switches**. I/O device utilization is maximized by **scheduling I/O-bound jobs as soon as they become ready to run**, thereby **incurring the overheads of context switches**. (1 credit) 1 credit per correct explanation (assuming 2 aspects mentioned)						

Nr:	6.6
U:	1/2013, Model

Q: Given the Java program fragment below:

```java
class Count extends Thread {
    public int inc;
    public Count(int inc) {
        this.inc = inc;
    }

    static int ctr = 0;

    public void run() {
        int loc;
        for(int i = 1; i <= 3; i++) {
            loc = ctr + inc;
            ctr = loc;
            System.out.println(inc + "\t" + loc);
        }
    }

    public static void main(String [] args) {
        Count p = new Count(1);
        Count q = new Count(-1);
        p.start();
        q.start();
        try { p.join(); q.join(); }
        catch(InterruptedException e) { }
        assert ctr == 0;
    }
}
```

(a) Give an example of the output of the program.

(b) What is wrong with the program and what do you think would have been the intention of the programmer for the output?

(c) Please indicate which declaration(s) and statement(s) should be added to the code to fix the problem.

C:	7 credits	T:	T	D:	8 min	L:	Medium

A:	(a) An example of the output of the program is (3 credits):
	`1      1`
	`-1     -1`
	`-1     -2`
	`1      2`
	`1      3`
	`-1     -3`
	(b) The program has a race condition. The programmer had intended the counter to be incremented 3 times and decremented three times, ending up as 0. (2 credits)
	(c) The semaphore declarations and statements to be added are: (2 credits)
	
	Before or after line 10:
	`static Semaphore s = new Semaphore(1);`
	Before line 15:
	`try { s.acquire(); }`
	`catch(InterruptedException e) {}`
	After line 16:
	`s.release();`

Nr:	7.3						
U:	1/2013, Model						
Q:	(a) List the three necessary conditions that must simultaneously hold for a deadlock to appear, in addition to mutual exclusion. (b) For each of the three conditions indicate what the specific problems are that each condition causes, and how the condition could be negated.						
C:	4 credits	T:	T	D:	5 min	L:	Moderate
A:	(a) The three conditions are (1 credit) • Hold and wait. • No pre-emption • Circular wait (b) The difficulties and possible remedies are: (1 credit each) • Hold and wait results in **long delays and inefficient use of resources**. One approach is to require processes to **request all resources in advance** (thus negating the wait but not the hold), which aggravates the delays and the inefficiency, but avoids deadlock. • No pre-emption applies to **some resources**, such as printers but not to others, such as the CPU, so for pre-emptible resources there is no problem. • Circular wait can be negated by **ordering the resource types**, so that all processes must request the resources in the same order. This results in **long waits** again.						

Nr:	10.9
U:	1/2013, Model

Q:	Consider the C program fragment below:
	
	`#define M 1024 /* max mount point file name size */`
	`char *lookup(char *pth, char *mnt) {`
	`    struct mntent m;`
	`    struct stat s;`
	`    stat(pth, &s);`
	`    dev_t d = s.st_dev;`
	`    ino_t i = s.st_ino;`
	`    FILE *f = setmntent("/proc/mounts", "r");`
	`    while (getmntent_r(f, &m, mnt, M)) {`
	`        if (stat(m.mnt_dir, &s) != 0) {`
	`            continue;`
	`        }`
	`        if (s.st_dev == d && s.st_ino == i) {`
	`            return mnt ;`
	`        }`
	`    }`
	`    endmntent(f);`
	`    return NULL;`
	`}`
	
	`int main(int argc, char **argv) {`
	`    char pth[M] = "/", mnt[M], *end;`
	`    strncat(pth,argv[1],M);`
	`    for(;;) {`
	`        if(lookup(pth,mnt) != NULL) {`
	`            printf("%s mounted on %s\n",pth,mnt);`
	`        } else {`
	`            printf("%s not mounted\n",pth);`
	`        }`
	`        end = strrchr(pth,'/');`
	`        if(end == NULL || end == pth) {`
	`            break;`
	`        } else {`
	`            *end = '\0';`
	`        }`
	`    }`
	`    return 0;`
	`}`
	(a) What is the output of the program when it is executed on the root directory?
	(b) What if the argument is a directory in a file system mounted somewhere on the root file system?						

C:	7 credits	T:	T	D:	10 min	L:	Medium

A:	(a) // mounted on rootfs
	/ mounted on rootfs (3 credit)
	(b) //home mounted on /dev/someting
	/ mounted on rootfs (4 credits)						

Nr:	11.8
U:	1/2013, Model

Q: Consider the C program fragment below:

```
#define M 1024
#define N 64
#define Dir "Foo"

int main(int argc, char* argv[]) {
    int i;
    char top[M], cur[M], tmp[M];
    getcwd(top,M);
    printf("top=%s\n",top);
    strcpy(cur, ".");
    for(i=0; i<N; i++) {
        strcpy(tmp, cur);
        sprintf(cur, "%s/Foo", tmp);
        mkdir(cur, 0755);
        sprintf(tmp, "%s/Foo_%d", top, i);
        symlink(cur, tmp);
    }
    return 0;
}
```

(a) How many directories are created by the program? Why?
(b) How many symbolic links are created by the program? Why?
(c) How many regular files does the program create? Why?

C:	7 credits	T:	T	D:	10 min	L:	Medium

A:

(a) The program creates 64 directories, each at one nesting level deeper than the previous. (3 credit)
(b) The program creates 64 symbolic links foo_i in the current directory, each pointing at a different subdirectory, where i indicates the depth of the directory. (2 credit)
(c) The program creates no ordinary files, because it never opens a file for writing. (2 credit)

Nr:	14.7
U:	1/2013, Model

Q: Consider the C program fragment below:

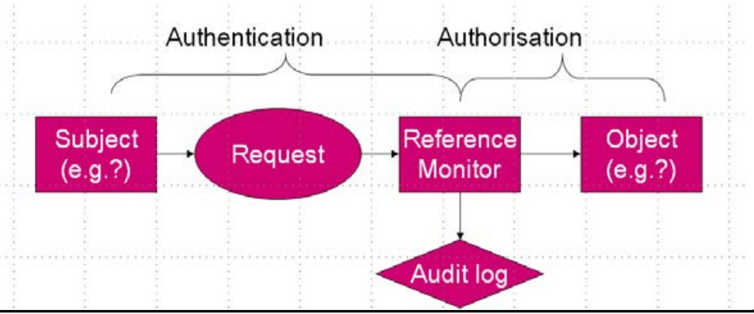
```
int main(int argc, char* argv[]) {
    struct passwd *p;
    while ((p = getpwent()) != NULL) {
        printf("%s:%s:%d:%d:%s:%s:%s\n",
           p->pw_name,
           p->pw_passwd,
           p->pw_uid,
           p->pw_gid,
           p->pw_gecos,
           p->pw_dir,
           p->pw_shell);
    }
    endpwent();
    return 0;
}
```

(a) How would a sysadmin normally produce the same output as the program?
(b) If two users happen to choose the same password on a modern Linux system, will this be visible? Why?

C:	4 credits	T:	T	D:	5 min	L:	Medium

A:
(a) `cat /etc/passwd`. (2 credits)
(b) The passwords in the password file are hashed with a **salt**, which ensures that even if two passwords are the same that the hashes are different. In any case most modern linux systems only use /etc/passwd during single-user modfe and not for normal operation. (2 credits)

Nr:	15.5						
U:	1/2013, Model						
Q:	In the context of access control: (a) What is authentication (b) What is authorization (c) What is auditing (d) What is the purpose of the reference monitor in relation to the three concepts mentioned above?						
C:	2 credits	T:	MC	D:	4 min	L:	Easy

A:	(a) Authentication is to determine who makes a request (0.5 credit) (b) Authorization is to determine who can do what on an object (0.5 credit) (c) Auditing is to determine post-hoc what happened and why (0.5 credit) (d) The reference monitor binds all three determinations together as follows (0.5 credit):  Authentication Authorisation Subject (e.g.?) Request Reference Monitor Object (e.g.?) Audit log