UNIVERSITEIT TWENTE.

Examination Operating Systems

8 April 2014

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 8:45 and ends at 12:15.

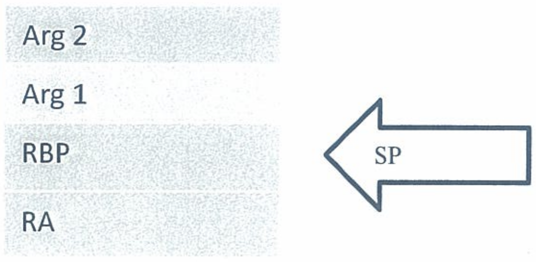
Try to give precise answers using appropriate terminology. For multiple-choice questions there may be more than one correct answer; all of these must be selected for full marks.

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.

Chapter 1:

Nr:	1.2						
Q:	A compiler generates code that pushes the arguments to a function onto the stack, such that the stack pointer always points to the location just before the first argument:  (a) Could this be a problem for code running in user space when an interrupt arrives? Why? (b) Could this be a problem for code running in kernel space when an interrupt arrives? Why?						
C:	4 credits						

Chapter 2:

Nr: 2.8

Q: Given the following C-program fragment:

```
#define N 41

int main(int argc, char * argv[]) {
    if(argc >= 3) {
        FILE *from = fopen(argv[1], "r");
        FILE *to = fopen(argv[2], "w");
        char buf[N];
        while (fgets(buf,N,from) != NULL) {
            fputs(buf,to); fputc('\n',to);
        }
        fclose(from);
        fclose(to);
        return 0;
    } else {
        printf("usage %s from to\n", argv[0]);
        return 1;
    }
}
```

(a) What is the output of the program when the input is a file with one line consisting of 120 characters + one newline character? (Hint: fgets reads everything, upto and including the next newline character into the buffer).
(b) Is it necessary to go to the disk for every gets call? Explain

C:	7 credits						

Chapter 4:

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?
(b) What is the output of the program? Explain.
(c) Is the output always the same? Explain.

C:	7 credits						

Chapter 5 (Silberschatz), 9 (Stallings):

Nr:	5.9

Q: Consider the C program fragment below:

```
#define N 8
#define M 1000000
typedef enum { False=0, True=1 } bool ;

void *tproc(void *ptr) {
    int k, i = *((int *) ptr);
    int bgn = sched_getcpu();
    printf("thread %d on CPU %d\n",i,bgn);
    for(k=0;k<M;k++) {
        int now = sched_getcpu();
        if( bgn != now ) {
            printf("thread %d moved to CPU %d\n",i,now);
            break;
        }
        sched_yield(); /* Schedule another thread */
    }
    pthread_exit(0);
}

int main(int argc, char * argv[]) {
    int i, targ[N];
    pthread_t thread[N];
    for(i=0; i < N; i++) {
        targ[i] = i;
        pthread_create(&thread[i], NULL,
            &tproc,(void *) &targ[i]);
    }
    for(i=0; i < N; i++) {
        pthread_join(thread[i], NULL);
    }
    return 0;
}
```

(a) Give an example of the output of the program on a dual core machine.
(b) Can a thread be run on different CPUs? If yes, what would the reason for the scheduler to move threads around? If no why not?
(c) Does Linux offer a library call to lock a thread to a specific CPU?

C:	7 credits							

Chapter 6 (Silberschatz), 5 (Stallings):

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) {                          while(true) {
 a1: Non_Critical_Section_1;            a2: Non_Critical_Section_2;
 b1: Wait(S);                           b2: Wait(S);
 c1: Critical_Section_1;                c2: Critical_Section_2;
 d1: Signal(S);                         d2: Signal(S);
}                                      }
```

C:	4 credits						

Nr: 6.9

Q: Given the Java monitor for the bounded buffer below:

```
 1   class Buffer{
 2       private int []B;
 3       private int Cnt = 0, In = 0, Out = 0;
 4
 5       Buffer(int size) {
 6          B = new int[size];
 7       }
 8
 9       public synchronized void Put(int i) {
10           while(Cnt == B.length) {
11               try{ wait(); }
12               catch(InterruptedException e) { }
13               finally{ }
14           }
15           B[In] = i;
16           In = (In + 1) % B.length;
17           Cnt++;
18           notify();
19       }
20
21       public synchronized int Get() {
22           while(Cnt == 0) {
23               try{ wait(); }
24               catch(InterruptedException e) { }
25               finally{ }
26           }
27           int i = B[Out];
28           Out = (Out + 1) % B.length;
29           Cnt--;
30           notify();
31           return i;
32       }
33   }
```

(a) What is the purpose of the while loops at lines 10-14 and 22-26?
(b) Why does C not offer monitors, like Java?

C: 7 credits

Chapter 7 (Silberschatz), 6 (Stallings):

Nr: 7.4

Q: Consider the C program fragment below:

```
 7  #define N 2
 8  #define P 3
 9  sem_t Room;
10  sem_t Fork[P];
11  void *tphilosopher(void *ptr) {
12      int i, k = *((int *) ptr);
13      for(i = 1; i <= N; i++) {
14          printf("%*cThink %d %d\n", k*4, ' ', k, i);
15          sem_wait(&Room) ;
16          sem_wait(&Fork[k]) ;
17          sem_wait(&Fork[(k+1) % P]) ;
18          printf("%*cEat %d %d\n", k*4, ' ', k, i);
19          sem_post(&Fork[k]) ;
20          sem_post(&Fork[(k+1) % P]) ;
21          sem_post(&Room) ;
22      }
23      pthread_exit(0);
24  }
25
26  int main(int argc, char * argv[]) {
27      int i, targ[P];
28      pthread_t thread[P];
29      sem_init(&Room, 0, P-1);
30      for(i=0;i<P;i++) {
31          sem_init(&Fork[i], 0, 1);
32      }
33      for(i=0;i<P;i++) {
34          targ[i] = i;
35          pthread_create(&thread[i], NULL,
36              &tphilosopher,(void *) &targ[i]);
37      }
38      for(i=0;i<P;i++) {
39          pthread_join(thread[i], NULL);
40      }
41      return 0;
42  }
```

(a) Give an example of the output of the program.
(b) What would happen if the Room semaphore was left out? Why?
(c) Does the order of the wait calls in lines 15-17 matter? Why?

C:	7 credits							

Nr: 7.5

Q: Given the simplified version of Dijkstra's Bankers Algorithm and associated state for a system with three processes and four resources below:

```
typedef enum { False=0, True=1 } bool ;

#define P 3
#define R 4

int reQuest[P][R] = {
              /*p0*/ {0,1,0,1},
              /*p1*/ {0,0,0,1},
              /*p2*/ {0,1,0,1} };

int Usage[P][R] =    {
              /*p0*/ {0,0,0,0},
              /*p1*/ {0,0,1,0},
              /*p2*/ {1,1,1,0} };

int Available[R] =   {0,0,0,1};

bool greater(int X[],int Y[]) {
    for(int j=0;j<R;j++) if(X[j]>Y[j]) return True ;
    return False ;
}

#define copy(X,Y)  for(int j=0;j<R;j++) X[j]=Y[j];
#define add(X,Y)   for(int j=0;j<R;j++) X[j]=X[j]+Y[j];

int main() {
    int Temp[R], Zero[R] = {0};
    copy(Temp,Available);
    for(int p=0; p<P; p++) {
        if(greater(Usage[p],Zero)) {
            if(greater(reQuest[p],Temp)){
                printf("deadlock %d\n",p);
            } else {
                printf("no deadlock %d\n",p);
                add(Temp,Usage[p]);
            }
        } else {
            printf("no deadlock %d\n",p);
        }
    }
    return 0;
}
```

(a) Which of the three processes p0 .. p2 are deadlocked? Explain.

(b) In what sense has the algorithm been simplified?

C:	7 credits						

Chapter 9 (Silberschatz), 8 (Stallings):

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						

Chapter 11 (Silberschatz), 12 (Stallings):

Nr:	11.7

Q: Consider the Linux shell script below (with line numbers added for ease of reference):

```
 1   /bin/rm a b c
 2   echo "Hello World" >a
 3   ln a b
 4   ls -i a b
 5   rm a
 6   cat b
 7   ln b a
 8   ls -i a b
 9   ln -s a c
10   cat c
11   /bin/rm a b
12   ls -i c
13   cat c
14   echo "Hello World" >a
15   ls -i a c
16   cat c
```

(a) What is the output of each of the four ls commands? (ls –i prints the inode and the filename)

(b) What is the output of each of the four cat commands?

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							

Chapter 14 (Silberschatz), 15 (Stallings):

Nr:	14.5						
Q:	(a) Why is an access control matrix typically sparse? (b) What is an access control list? (c) Why does the system have to manage capabilities? (Hint: what would go wrong if users could manipulate capabilities?)						
C:	3 credits						

Chapter 15 (Silberschatz), 14 (Stallings):

Nr:	15.6							
Q:	(a) Define confidentiality (b) Define integrity (c) Define availability (d) These three terms together are usually referred to as the .... ?							
C:	2 credits							

Nr: 15.7

Q: Consider the C program fragment below (due to Ken Thompson, 1984):

```
char s[ ] = {
        '\t',
        '0',
        '\n',
        '}',
        ';',
                        ← 220 lines deleted here
        0
};

/*
 * The string s is a
 * representation of the body
 * of this program from '0'
 * to the end.
 */

main( )
{
        int i;

        printf("char\ts[ ] = {\n");
        for(i=0; s[i]; i++)
                printf("\t%d, \n", s[i]);
        printf("%s",s);
}
```

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

C:	7 credits							

Nr: 15.8

Q: Consider the C program fragment below:

```
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

Lab 2014

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							