Operating Systems
Jacobs University Bremen
Dr. Jürgen Schönwälder

Course: 320202
Date: 2010-05-26
Duration: 120 minutes

Final Examination

The Jacobs University's Code of Academic Integrity applies to this examination. Please fill in your name (please write readable) and sign below.

Name:	
Signature:	

This exam is **closed book**. In addition, you are not allowed to use any electronic equipment such as computers, smart phones, cell phones, or calculators.

Please answer the questions on the problem sheets. If you need more space, feel free to write on the back of the pages. Please keep the papers stapled.

Problem	Max. Points	Points	Grader
F.1	10		
F.2	10		
F.3	20		
F.4	12		
F.5	8		
F.6	20		
F.7	10		
F.8	10		
Total	100		

Good luck!

Problem F.1: *deadlocks and scheduling* (2+2+2+2+2 = 10 points)

Indicate which of the following statements are correct or incorrect by marking the appropriate boxes. For every correctly marked box, you will earn two points. For every incorrectly marked box, you will loose one point. Statements which are not marked or which are marked as true and false will be ignored. The minimum number of points you can achieve is zero.

true	false	
☐	☐	All necessary conditions for a deadlock are mutual exclusion, hold and wait, and circular wait.
☐	☐	A cycle in a resource allocation graph is sufficient condition for a deadlock to occur.
☐	☐	The banker's algorithm at each iteration finishes a random process who's maximum resource requests can be satisfied.
☐	☐	In a safe state, all processes can at some point in time obtain their maximum resource requests.
☐	☐	Unix/Linux systems by default use probabilistic multilevel feedback queue scheduler to schedule user processes.

Solution:

true	false	
☐	☒	All necessary conditions for a deadlock are mutual exclusion, hold and wait, and circular wait.
☐	☒	A cycle in a resource allocation graph is sufficient condition for a deadlock to occur.
☒	☐	The banker's algorithm at each iteration finishes a random process who's maximum resource requests can be satisfied.
☒	☐	In a safe state, all processes can at some point in time obtain their maximum resource requests.
☒	☐	Unix/Linux systems by default use probabilistic multilevel feedback queue scheduler to schedule user processes.

Problem F.2: *processes* (10 points)

Which function does the following C program calculate? Explain how the calculation is carried out.

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

static int f(int n)
{
    pid_t p1, p2;
    int r1, r2;

    if (n > 1) {
        if (! (p1 = fork())) exit(f(n-2));
        if (! (p2 = fork())) exit(f(n-1));
        waitpid(p1, &r1, 0), waitpid(p2, &r2, 0);
        n = (WEXITSTATUS(r1) + WEXITSTATUS(r2));
    }
    return n;
}

int main(int argc, char **argv)
{
    for (int i = 1; argv[i]; i++) {
        printf("f(%d) = %d\n", atoi(argv[i]), f(atoi(argv[i])));
    }
    return 0;
}
```

Hint: The macro `WEXITSTATUS` extracts the return code from the number returned in the second argument of `waitpid()` and `atoi()` converts a character string into a number.

Solution:

The program prints for every number x passed on the command line the string "f(x) = y", where $y = f(x)$ is defined as follows:

$$f(x) = \begin{cases} x & x \leq 1 \\ f(x-2) + f(x-1) & x > 1 \end{cases}$$

For $x \geq 0$, the numbers $f(x)$ are known as Fibonacci numbers. The program calculates the Fibonacci sequence by executing every (recursive) call to $f(x)$ with $x > 1$ in separate child processes.

Problem F.3: *bus loading problem* (20 points)

Busses arrive and depart at a bus terminal. Only the first bus in a potential queue of busses loads passengers and it departs from the terminal once it is fully loaded. Each bus can hold a total of $N = 6$ passengers. $W = 2$ seats are reserved for wheelchair passengers and the remaining $R = 4$ seats are reserved for regular passengers. A bus may depart from the terminal only once all free spaces have been occupied. Note that passengers (regular or wheelchair) arrive and board in random order.

The following pseudo code implementing an outline of a solution for the bus and passenger threads is provided for your convenience. Fill in the missing semaphore operations in the `bus()` and `wheelchair_passenger()` and `regular_passenger` functions.

```
const int W = 2;
const int R = 4;
const int N = W + R;
semaphore terminal = 1;
semaphore wheelchairs = 0;
semaphore regularchairs = 0;
semaphore handshake = 0;

void bus()
{
    arriving();









    leaving();
}

void wheelchair_passenger()
{



    boarding();



}

void regular_passenger()
{



    boarding();



}
```

You can get 5 bonus points if you provide an additional solution that allows a bus to depart with some empty wheelchair seats if there are no waiting wheelchair passengers.

Solution:

```
const int W = 2;
const int R = 4;
const int N = W + R;
semaphore terminal = 1;
semaphore wheelchairs = 0;
semaphore regularchairs = 0;
semaphore handshake = 0;

void bus()
{
    arriving();
    down(&terminal);
    for (i = 0; i < W; i++) up(&wheelchairs);
    for (i = 0; i < R; i++) up(&regularchairs);
    for (i = 0; i < N; i++) down(&handshake);
    up(&terminal)
    leaving();
}

void wheelchair_passenger()
{
    down(&wheelchairs);
    boarding();
    up(&handshake);
}

void regular_passenger()
{
    down(&regularchairs);
    boarding();
    up(&handshake);
}
```

Problem F.4: *segment positioning strategies* (4+4+4 = 12 points)

Consider a system with 1024K of memory as shown below:

Process	Memory Area	Size [K]
	0-63	64
P3	64-191	128
	192-447	256
P1	448-575	128
	576-639	64
P2	640-895	256
	896-1023	128

The processes P1, P2 and P3 are already in memory when the processes P4, P5, P6 and P7 arrive in this order. P4, P5, P6 and P7 are of size 64K, 64K, 128K and 32K respectively. Assume that P3 was the last process loaded into memory before P4, P5, P6 and P7 arrive. Draw tables, similar to the one above, to show how these processes would be placed in memory by the following positioning algorithms:

a) First-fit

b) Next-fit

c) Best-fit

If a process does not fit into memory, label it as out of memory.

Solution:

First-fit		
Process	Memory Area	Size [K]
P4	0-63	64
P3	64-191	128
P5	192-255	64
P6	256-383	128
P7	384-415	32
	416-447	32
P1	448-575	128
	576-639	64
P2	640-895	256
	896-1023	128

Next-fit		
Process	Memory Area	Size [K]
	0-63	64
P3	64-191	128
P4	192-255	64
P5	256-319	64
P6	320-447	128
P1	448-575	128
P7	576-607	32
	608-639	32
P2	640-895	256
	896-1023	128

Best-fit		
Process	Memory Area	Size [K]
P4	0-63	64
P3	64-191	128
P7	192-223	32
	224-447	224
P1	448-575	128
P5	576-639	64
P2	640-895	256
P6	896-1023	128

Problem F.5: *page replacement strategies* (4+4 = 8 points)

Consider the following page-reference string:

$$w = 0, 4, 2, 5, 6, 0, 1, 3, 2, 4, 7, 1, 2, 1, 7, 1$$

Supposed there are four page frames, all initially unusued. Draw the memory tables for different page replacement algorithms, as shown below, and indicate with a $\star$ where the page faults occur. The following pages have already been loaded:

page access	frame #1	frame #2	frame #3	frame #4	fault
0	0				$\star$
1	0	1			$\star$
2	0	1	2		$\star$
3	0	1	2	3	$\star$

In case of a tie, replace the page with the larges page number. Draw a separate table for each of the following algorithms:

a) FIFO (first in first out)

b) LRU (least recently used)

Solution:

FIFO					
page	fr. #1	fr. #2	fr. #3	fr. #4	fault
0	0				$\star$
1	0	1			$\star$
2	0	1	2		$\star$
3	0	1	2	3	$\star$
0	0	1	2	3	
4	4	1	2	3	$\star$
2	4	1	2	3	
5	4	5	2	3	$\star$
6	4	5	6	3	$\star$
0	4	5	6	0	$\star$
1	1	5	6	0	$\star$
3	1	3	6	0	$\star$
2	1	3	2	0	$\star$
4	1	3	2	4	$\star$
7	7	3	2	4	$\star$
1	7	1	2	4	$\star$
2	7	1	2	4	
1	7	1	2	4	
7	7	1	2	4	
1	7	1	2	4	

LRU					
page	fr. #1	fr. #2	fr. #3	fr. #4	fault
0	0				$\star$
1	0	1			$\star$
2	0	1	2		$\star$
3	0	1	2	3	$\star$
0	0	1	2	3	
4	0	4	2	3	$\star$
2	0	4	2	3	
5	0	4	2	5	$\star$
6	6	4	2	5	$\star$
0	6	0	2	5	$\star$
1	6	0	1	5	$\star$
3	6	0	1	3	$\star$
2	2	0	1	3	$\star$
4	2	4	1	3	$\star$
7	2	4	7	3	$\star$
1	2	4	7	1	$\star$
2	2	4	7	1	
1	2	4	7	1	
7	2	4	7	1	
1	2	4	7	1	

Problem F.6: *file system organization* (2+2+2+2+2+2+6+2 = 20 points)

a) Define the terms index node (inode). What are inodes used for?

b) Define the term data block. What are data blocks used for?

c) Define the term super block. What are super blocks used for?

d) Define the term directory. What does a directory contain and how are directories stored?

e) Where is meta-data of a file (e.g., the size of a file or the last modification time stamp) stored?

f) Explain the difference between a hard link and a soft link.

g) Consider the following sequence of shell commands executed in an empty directory:

```
echo 1 > a
ln -s b c
echo 2 > c
ln c d
cat d
```

What is the content of the current directory after execution of these shell commands? Make a drawing showing the directory, its content, and any inodes that might be involved. What will `cat` display on the standard output? (Note: `ln -s` creates a symbolic link, `ln` without the `-s` option creates a hard link.)

h) Device files are an interface for a device driver that appears in a file system. How does a device file identify the underlying device and its device driver?

Solution:

a) An inode represents a file or part of a file. The inode of a file contains data (if it is short) or pointers to data nodes or pointers to inodes that contain pointers to further data nodes.

b) Data blocks contain a block of data of a file. They are referenced from inodes.

c) A super block contains meta-data about a file system (e.g, its size). The meta-data includes among other things the inode number of the root directory of the file system.

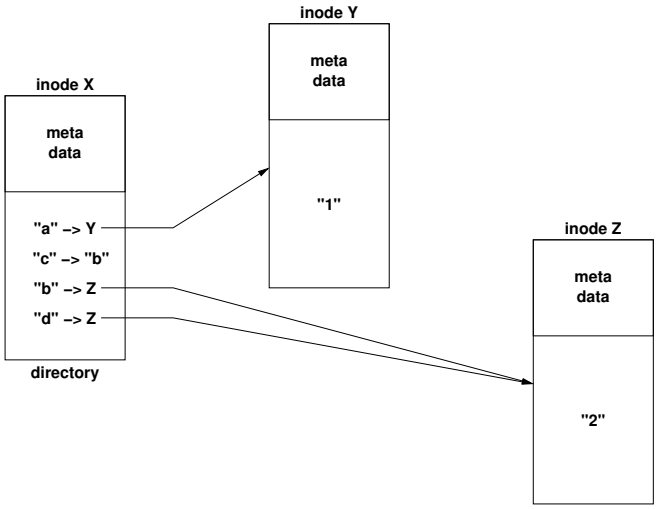
d) A directory is a special file that maintains a mapping of file names (contained in the directory) to inode numbers.

e) The meta-data of a file is stored in the inode representing the file.

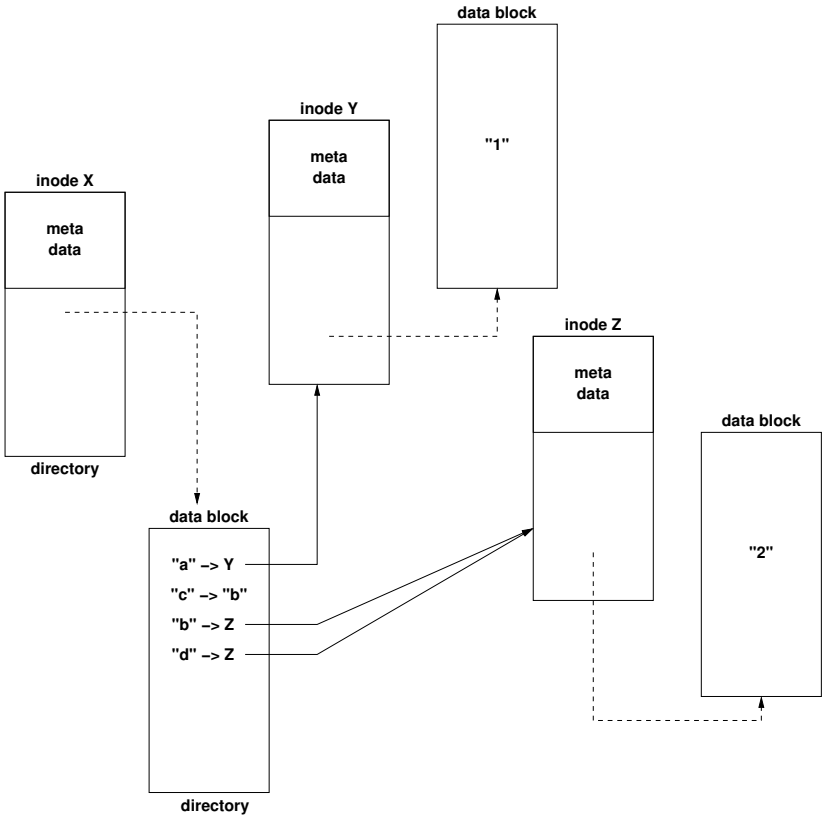
f) A hard-link for a name in a directory pointing to the inode number of the original file. In other words, a hard-link is a second name for a given file. A symbolic-link is a name in a directory refering to a file system path that should be used if the directory name is opened.

g) The first line writes “1” into a new file `a`. The second line creates a symbolic link `c` pointing to the currently non-existent file `b`. The third line writes “2” to `c`, which causes the file `b` to be created. The fourth line creates a hard link from `c` to `d`. Since `c` is a symbolic link, this results in a hard link for the regular file `b`. Finally, `cat` will display “2” on the standard output. As a consequence, the directory will contain the regular files `a`, `b`, `d` and the symbolic link `c`. The files `b` and `d` both refer to the same file content (inode).

The first drawing shows the situation with data inlined into inodes.



The second drawing shows the situation with no data inlining.



h) A device file has a major and a minor device number. The major device number identifies a device driver in the kernel and the minor device number identifies a particular device (possibly out of many) that the driver controls.

Problem F.7: *input/output and devices* (2+2+3+3 = 10 points)

a) Briefly explain the difference between programmed I/O and interrupt driven I/O.

b) Explain the difference between a block device and a character device. Provide an example for each device type.

c) Explain the term memory mapped file I/O. What are the benefits of memory mapped file I/O? Provide an example where memory mapped file I/O is commonly used.

d) Explain the term vectored I/O, also known as scatter/gather I/O. What are the benefits of vectored I/O? Provide an example where vectored I/O is commonly used.

Solution:

a) With programmed I/O, the CPU does all the work (copying data to/from the device) and it can't do other work until I/O is complete. With interrupt-driven I/O, interrupts drive the I/O process and the CPU can do other things while the device is busy.

b) The basic unit of work of block devices is a fixed-length data block. The basic unit of work of character devices are characters, often encoded in octets. Devices representing disk partitions are block devices while devices representing terminals are character devices.

c) Memory mapped file I/O maps the content of files into the logical address spaces of running processes. Memory mapped file I/O is very efficient, providing efficient random access to a file's content. In addition, multiple processes may share memory mappings.

   Memory mapped file I/O is commonly used to load dynamically linked shared libraries into a process image, making use of the feature to share mapped files between multiple processes.

d) Vectored I/O is an I/O method in which a single system call either writes data from multiple buffers to a single data stream or reads data from a single data stream to multiple buffers. Vectored I/O is highly efficient if data already exists in multiple data chunks because the total number of system calls needed can be minimized and unnecessary copying of data can be avoided.

   Vectored I/O is commonly used when a complex data structure consisting of several headers and the itself data needs to be read/written. In such a case, it is often easier to prepared the various headers in separate memory buffers and to write the I/O vector in a single system call. Similarly, if the kernel receives multiple chunks of data from a device, it can pass all the chunks up to user space via a single vectored I/O system call.

Problem F.8: *signals, pipes, sockets* (2+2+2+2+2 = 10 points)

Indicate which of the following statements are correct or incorrect by marking the appropriate boxes. For every correctly marked box, you will earn two points. For every incorrectly marked box, you will loose one point. Statements which are not marked or which are marked as true and false will be ignored. The minimum number of points you can achieve is zero.

true	false	
☐	☐	A floating point exception signal is a synchronous signal while an alarm signal is an asynchronous signal.
☐	☐	The `sigsuspend()` system call suspends the calling process until delivery of a signal whose action is to invoke a signal handler or to terminate the process.
☐	☐	Pipes created using the `pipe()` system call are unidirectional byte streams used between a parent process and a child process.
☐	☐	Stream sockets (`SOCK_STREAM`) and datagram sockets (`SOCK_DGRAM`) use different address families and thus different socket address structures to name a socket.
☐	☐	A server with a single listening socket may use the `select()` system call to decide which client to accept.

Solution:

true	false	
☒	☐	A floating point exception signal is a synchronous signal while an alarm signal is an asynchronous signal.
☒	☐	The `sigsuspend()` system call suspends the calling process until delivery of a signal whose action is to invoke a signal handler or to terminate the process.
☒	☐	Pipes created using the `pipe()` system call are unidirectional byte streams used between a parent process and a child process.
☐	☒	Stream sockets (`SOCK_STREAM`) and datagram sockets (`SOCK_DGRAM`) use different address families and thus different socket address structures to name a socket.
☐	☒	A server with a single listening socket may use the `select()` system call to decide which client to accept.