

Module Name: Operating Systems

Module Number: CE01000-3

Title of Assessment: Portfolio of exercises

Module Learning Outcomes for This Assessment

3. Apply to the solution of a range of problems, the fundamental concepts, principles and algorithms employed in the operation of a multi-user/multi-tasking operating systems.	

Hand in deadline: Friday 26th November, 2010.

Assessment description

Selected exercises from the weekly tutorial/practical class exercises are to be included in a portfolio of exercises to be submitted Faculty Office in a folder by the end of week 12 (Friday 26th November 2010).

The weekly tutorial/practical exercises that are to be included in the portfolio are specified below.

What you are required to do.

Write up the selected weekly exercises in Word or other suitable word processor.

The answers you submit should be your own work and not the work of any other person.

Note the University policy on **Plagiarism and Academic Dishonesty** - see *Breaches in Assessment Regulations: Academic Dishonesty* - at <http://www.staffs.ac.uk/current/regulations/academic/index.php>

Marks

The marks associated with each selected exercise will be indicated when you are told which exercises are being selected.

Week 1.

Tutorial questions:

6. How does the distinction between supervisor mode and user mode function as a rudimentary form of protection (security) system?

The operating system assures itself a total control of the system by establishing a set of privileged instructions only executable in supervisor mode. In that way the standard user won't be able to execute these commands, thus the security.

7. What are the differences between a software interrupt (trap) and a hardware interrupt? What is the use of each?

Traps (also named programmed exceptions or software interrupt), are a type of exceptions caused, intentionally or not, by another program under the form of an error.

An example would be the division by zero, this division caused the program to abort the wanted operation, the program knows a result will never be reached.

Traps are also use when a process in queue has a higher priority than the one being processed, allowing the appropriate process to access the CPU. The system calls uses interrupt.

Hardware interrupt happens when a piece of hardware has failed due to a problem or error and need to stop functioning in order to avoid further damage or has to be fixed. Power shut down, overheat, device plugged in... are examples.

Week 2.

Tutorial question:

4. What is the purpose of system calls? Why are they necessary?

System calls provides an interface to communicate with the operating system kernel.

The Kernel is the core of the OS, it manages the hardware resources.

The kernel provides an interface to the application programs in the form of system calls(i.e. access to file, between process communication is provided via system calls)

When a program launches a system call, the information currently held by the programmed is stored and saved before this one is being shut down. The information saved will be used in the restoration of the program.

The CPU starts executing a series of command shaped by the systems calls in kernel mode before returning command to the user once the program is restored.

Week 3.

Virtual machine exercise:

16) Writing a short Shell script

Use man to find out what the head command does.

Using an editor, write a short (two or four line) shell script which applies sort and head to a test file and will give a sorted list of the first five lines in the file.

```
sort -otest < test  
head -n5 < test
```

Tutorial question:

12) The ability of a process to create a new process is an important capability, but it has dangers. Consider the consequences of allowing a user to run the process below. Assume that fork() is a system call that creates a new child process.

```
int main() {  
  
    while (true) {  
        fork();  
    }  
}
```

a) What would the consequences of allowing such a process to run? (2 marks)

This program is called a Fork bomb. A fork bomb creates a large number of processes very quickly saturating the space available to other process kept in queue by the OS. The process table eventually gets saturated, new programs won't be able to start until the last terminates. It is unlikely that any other program will lunch since the instance of the Fork bomb will on each attempt try to take any newly available slot. Moreover each child process of the program will use more processor time and memory thus slowing down the entire system and other programs to the point of becoming unusable.

b) Suppose that you are an operating systems designer and have been asked to build in safeguards against such processes. Assume that you have decided that it is inappropriate to reject certain processes, and that the best approach is to place certain runtime controls on them. What controls might the operating system use to detect processes like the above at runtime?

A Way to prevent such a program to saturate the process table is by limiting the number of processes that a single user may own.

When such a process would try to create another process, and if the maximum process that the owner can own is reached, the creation of the child fails. The maximum number of processes that can be own is to be set by administrator, as low enough so that simultaneous launch of such a program by different user would allowed the system not to crash and kill the unwilled processes.

Week 4.

Tutorial questions:

6) Consider the following set of processes, with the length of the CPU-burst time given in milliseconds:

Process	Burst Time	Priority
P1	4	2
P2	1	1
P3	3	3
P4	1	4
P5	7	3

At time 0, the processes are queued in the order of arrival P1, P2, P3, P4, P5.

FCFS (First Come First Serve Scheduling)

P1	P2	P3	P4	P5
0	4	5	8	9
				16

SJF (Shortest-Job-First Scheduling)

P2	P4	P3	P1	P5
0	1	2	5	9
				16

Non Preemptive Scheduling

P2	P1	P5	P3	P4
0	1	5	12	15
				16

RR (Round-Robin Scheduling, quantum = 1)

P1	P2	P3	P4	P5	P1	P3	P5	P1	P3	P5	P1	P5
0	1	2	3	4	5	6	7	8	9	10	11	12
												16

What is the turnaround time of each process for each of the scheduling algorithms in part a?

Turnaround Time	P1	P2	P3	P4	P5
FCFS	4	5	8	9	16
SJF	9	1	5	2	16
NPP	5	1	15	16	12

RR Q=1	12	2	10	4	16
--------	----	---	----	---	----

What is the waiting time of each process for each of the scheduling algorithms in part a?

Waiting Time	P1	P2	P3	P4	P5
FCFS	0	4	5	8	9
SJF	5	0	2	1	9
NPP	1	0	12	15	5
RR Q=1	8	1	7	3	9

Which of the schedules in part a results in the minimal average waiting time (over all processes)?

$$\text{FCFS} = \frac{4+5+8+9}{5} = 5.2$$

$$\text{SJF} = \frac{5+2+1+9}{5} = 3.4$$

$$\text{NPP} = \frac{1+12+15+5}{5} = 6.6$$

$$\text{RR} = \frac{8+1+7+3+9}{5} = 5.6$$

The minimal result is obtained by SJF.

9) Determining the quantum is a complex and critical task. Assume that the average context-switching time between processes is s , and the average amount of time an I/O bound processes uses before generating an I/O request is t ($t \gg s$). Discuss the effect of each of the following quantum settings, q .

a) q is slightly greater than zero

The quantum is too low; process will be executed for a too short period of time. The spent on switching between the different processes will be great. Overall the CPU will loose time.

b) $q = s$

Again the time q is too small for the process to be started or continued.

c) $s < q < t$

d) $q = t$

e) $q > t$

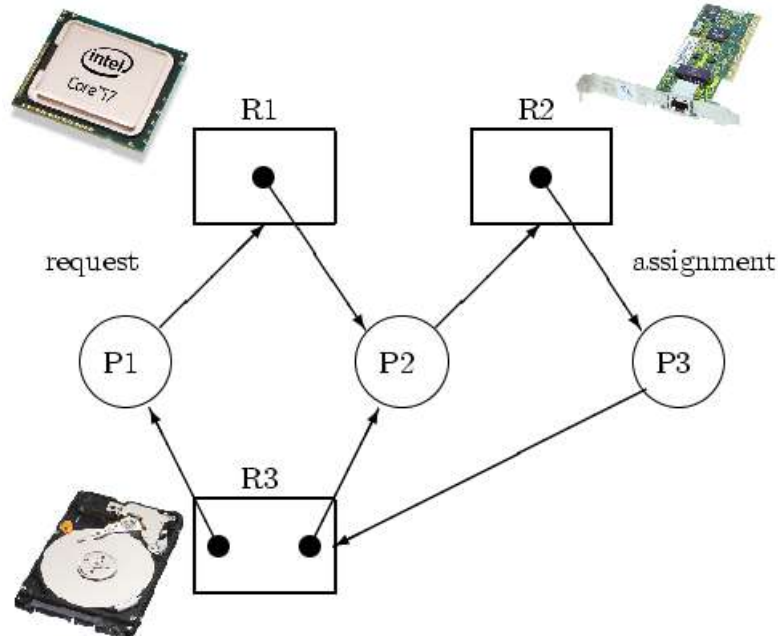
f) q is an extremely large number

(12 marks)

Week 5.

Tutorial question:

8) Give an example of a simple resource deadlock involving 3 processes and 3 resources. Draw the Resource Allocation Graph that illustrates this.



This is a Simple example of a deadlock, the first process (P1) requests the CPU as a resource, while the hard drive is being assigned to it, P2 as both the CPU and one hard drive assigned to it but is waiting the network chip and P3 has the network chip but is requesting the Hard drive.

All resources are unavailable for request as they are already assigned to another process. The system is in a deadlock state.

Virtual machine exercises with Linux processes.

17) Use your forktest.c program as the basis, but modify it by deleting the printf's that occur under the if and the else that output the PIDs and replace them with code for the parent to output the series of odd (in French 'impair') numbers in an infinite loop (e.g. 1, 3, 5, 7, 9,...) and for the child process to output the series of even (in French 'pair') numbers in an infinite loop (e.g. 2, 4, 6, 8, 10,..). You could do this by using a for loop under the relevant branch of the if..else statement.

(6 marks)

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

main()
{
    pid_t val;
    int ODD = 1, EVEN = 0, n;

    printf("PID before fork(): %d\n", (int)getpid());
    if(val=fork())          //process 1 : Parent
    {
        printf("PID after fork(): %d - The parent\n", (int)getpid());
        for(;;)//Infinite loop because the CPU goes too fast, we now see "Parent"
        {
            printf("Parent\n");
        }
    }
    else                    //process 2 : Child
    {
        printf("PID after fork(): %d - The child\n", (int)getpid());
        for(;;)//Infinite loop because the CPU goes too fast, we now see "Child" after
        {
            printf("Child\n");
        }
    }
}
```

23) Up till now the parent process has continued to execute while the child process has executed. We can use the wait() system call to make the parent process suspend itself and wait until it is woken up by the termination of the child process.

So modify forktest4.c to make the parent process wait for the child.

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

main()
{
    pid_t val;
    int ODD = 1, EVEN = 0, status;

    printf("PID before fork(): %d\n", (int) getpid());
    if(val=fork())          //process 1 : Parent
    {
        printf("PID after fork(): %d - The parent\n", (int) getpid());
        wait(&status);
    }
    else                    //process 2 : Child
    {
        printf("PID after fork(): %d - The child\n", (int) getpid());
        execl("/bin/ls", "ls", "-l", 0);
        status;
    }
}
```

Week 6.

Tutorial question:

2. One method of deadlock prevention by denying the 'wait for' condition requires that processes must request all of the resources they will need before the system will allow them to compete for resources. The system grants resources on an 'all or none' basis. Discuss the advantages and disadvantages of this method.

This method requires that all of the resources needed by the process to be executed have to be requested at once. The system must therefore assign resources on an all or none base.

If all of the resources needed by the process are not currently available, the process will wait until the set of resources required is available.

The good side is that while waiting, the process won't hold out any resources, thus avoiding any possibility of a deadlock state.

The down side of this “wait for” strategy is that it will surely lead to a great waste of resources.

As a example, a program requiring 10 tap drives will have to request and receive all 10 drives, if this program needs only one tap drive to execute, and doesn’t need the other 9 drives for several hours, these 9 drives will remain unusable for some time. This method can lead to indefinite postponement, or starvation. Since not all the required resources may never be available at the same time.

Virtual machine exercises – simple windows system programming.

14) Now we are in a position to modify the program above. We are going to change it so that now instead of copying a file from one file to another, the code will copy a file but output it to the console window. So it is a little like the DOS Type command.

```
#include <windows.h>
#include <stdio.h>
#define BUF_SIZE 2048
int main (int argc, LPTSTR argv[])
{
    HANDLE hIn, hOut;
    DWORD nIn, nOut;
    CHAR Buffer [BUF_SIZE];
    if(argc !=2)
    {
        printf("Usage: cp file1 file2\n");
        return 1;
    }
    hIn = CreateFile (argv[1],
    GENERIC_READ,FILE_SHARE_READ,NULL,OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL,NULL);
    if(hIn==INVALID_HANDLE_VALUE)
    {
        printf("Cannot open input file. Error: %x\n", GetLastError());
        return 2;
    }
    while (ReadFile(hIn,Buffer,BUF_SIZE,&nIn,NULL)&&nIn>0)
    {
        hIn=GetStdHandle(STD_OUTPUT_HANDLE);
        WriteFile(hIn, Buffer, nIn, &nOut, NULL);
        if(nIn != nOut)
        {
            printf("Fatal write error:%x\n",GetLastError());
            return 4;
        }
    }
    CloseHandle(hIn);
    CloseHandle(hOut);
    return 0;
}
```

Week 7.

Tutorial questions

4) Given memory partitions of 200K, 450K, 100K, 700K, and 300K (in order), how would each of the First-fit, Best-fit, and Worst-fit algorithms place processes of 212K, 417K, 112K, and 426K (in order)? Which algorithm makes the most efficient use of memory?

	First Fit	Best Fit	Worst Fit
200K	200	112 88	200
450K	212 238	417 33	417 83
100K	100	100	100
700K	417 283	426 212 62	212 112 376
300K	112 188	300	300
Wait Queue	426		426
Memory Left	1009	583	1059

Virtual machine exercise:
Linux interprocess communication.

14) Use pipes to communicate between the parent and child process.

Remember the parent process repeatedly writes the alphabet to the pipe using an infinite loop and the child process repeatedly reads from the pipe and actually outputting the characters to the screen.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#define SIZE 26          //Alphabet

main()
{
    int i,count,fda[2];
    char buf [SIZE];
    char ch;

    if (pipe(fda)==-1)          //Share the data
        printf("error creating pipe\n");

    if (fork()) //Process 1 : Parent
    {
        close (fda[0]);
        for(i=0;i<26;i++)
        {
            buf[i]=i+97;          //Letters
        }

        if(write(fda[1],buf,SIZE)!=SIZE)
            printf("error writing to pipe\n");
    }
    else //Process 2 : Child
    {
        for(;;)          //Infinite Loop
        {
            close (fda[1]);
            if (read(fda[0],buf,SIZE)==-1) //Get fda via buffer
                printf("error reading pipe\n");
            for(i=0;i<26;i++)
            {
                printf("%c",buf[i]);
            }
        }
    }
}
```

Week 8.

Tutorial question.

4) Consider the following page reference string:

1, 2, 3, 2, 4, 2, 5, 2, 3, 6, 2, 5, 1, 2, 3

How many page faults would occur for the following replacement algorithms, assuming three and four frames?

- LRU replacement

- FIFO replacement

- Optimal replacement

LRU	1	2	3	2	4	2	5	2	3	6	2	5	1	2	3	Page Faults
3 Frame	1	1	1	1	4	4	4	4	3	3	3	5	5	5	3	
		2	2	2	2	2	2	2	2	2	2	2	2	2	2	
			3	3	3	3	5	5	5	6	6	6	1	1	1	
Page Faults	x	x	x		x		x		x	x		x	x		x	10

4 Frame	1	1	1	1	1	1	5	5	5	5	5	5	1	1	1	
		2	2	2	2	2	2	2	2	2	2	2	2	2	2	
			3	3	3	3	3	3	3	3	3	3	3	3	3	
				4	4	4	4	4	6	6	6	6	6	6	6	
	x	x	x		x		x		x		x					7

FIFO	1	2	3	2	4	2	5	2	3	6	2	5	1	2	3	Page Faults
3 Frame	1	1	1	1	4	4	4	4	3	3	3	3	1	1	1	
		2	2	2	2	2	5	5	5	6	6	6	6	2	2	
			3	3	3	3	3	2	2	2	2	5	5	5	3	
Page Faults	x	x	x		x		x	x	x	x		x	x	x	x	12

4 Frame	1	1	1	1	1	1	5	5	5	5	5	5	5	5	3	
		2	2	2	2	2	2	2	2	6	6	6	6	6	6	
			3	3	3	3	3	3	3	3	3	2	2	2	2	
				4	4	4	4	4	4	4	4	4	1	1	1	
Page Faults	x	x	x		x		x		x	x		x	x		x	9

Optimal	1	2	3	2	4	2	5	2	3	6	2	5	1	2	3	Page Faults
3 Frame	1	1	1	1	4	4	5	5	5	5	5	5	1	1	1	
		2	2	2	2	2	2	2	2	2	2	2	2	2	2	
			3	3	3	3	3	3	3	6	6	6	6	6	3	
Page Faults	x	x	x		x		x		x		x		x		x	8

4 Frame	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
		2	2	2	2	2	2	2	2	2	2	2	2	2	2	
			3	3	3	3	3	3	3	6	6	6	6	6	6	
				4	4	5	5	5	5	5	5	5	5	5	3	
Page Faults	x	x	x		x		x		x				x			7

Week 9.

Tutorial questions.

3. Assume a system uses 2K blocks and 16 bit (2 byte) addresses. What is the largest file size that such a system can support using indexed block file allocation with a single level index. Comment on the resulting maximum size in terms of its adequacy for a modern filing system.

9. What problems might arise when a file is deleted, if it is shared?

If nothing is done to prevent a file from being deleted, users will be able to delete the original file. The file will therefore be unavailable to others users.

10. How can we solve this problem?

A copy or a pointer of/on the original file undeletable by any user would let the file available to everyone.

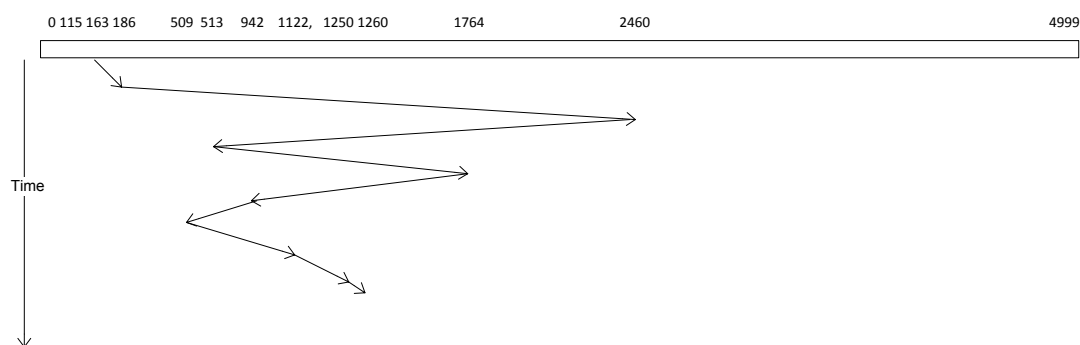
Counting the number of deleting calls being made on a file would be a good way verify that people really want to delete the file, when the counter reach zero the file is then willingly deleted by the system.

Week 10.

1) Suppose that a disk drive has 5000 cylinders, numbered 0 to 4999. The drive is currently serving a request at cylinder 163, and the previous request was at cylinder 115. The queue of pending requests, in FIFO order, is 186, 2460, 513, 1764, 942, 509, 1122, 1250, 1260.

Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests, for each of the following disk scheduling algorithms?

a. FCFS

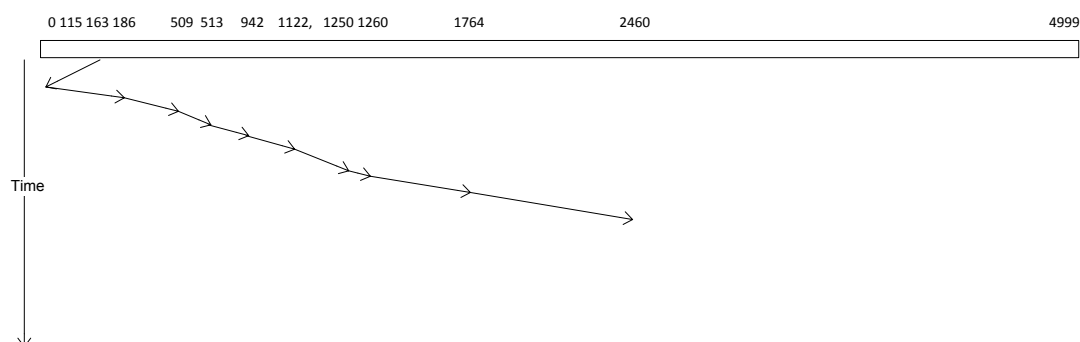


Queue is 186, 2460, 513, 1764, 942, 509, 1122, 1250, 1260; With a head start at 163.

$$(186-163)+(2460-186)+(2460-513)+(1764-513)+(1764-942)+(942-509)+(1122-509)+(1250-1122)+(1260-1250)= 7501$$

7501 is the total head movement done to satisfy the pending requests using the First Come First Served Algorithm

b. SCAN



$$163+186+(509-186)+(513-509)+(942-513)+(1122-942)+(1250-1122)+(1260-1250)+(1764-1260)+(2460-1764)= 2623$$

2623 is the total head movement done to satisfy the pending requests using the SCAN (or elevator algorithm).