

Operating systems Portfolio

WEEK 1

Questions

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

Supervisor mode has access to launching all applications, user mode is a restricted mode with only few applications available, which assure the operating system cannot be affected.

These two modes assure a good deal to the system, supervisor has access to some memory blocks crucial.

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

A hardware interrupt is launched by devices when Input/Output completion exists or for signalling a problem with a device.

A software interrupt is executed by a program for signalling to the operating system that its intervention is needed (the software has made an error or something else).

WEEK 2

Questions

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

System calls are a method (or interface) for communicating with the kernel in low-level language.

The kernel is the part of the operating system which controls hardware, without a system call, an application cannot have access to hardware, and for example can't read or write a file.

When a program uses a system call, this program is paused for a time to let the CPU to execute the system call.

System calls are used by a program to communicate with the operating system by the help of a simple language called «library functions».

System calls represent a basic «must-have» interface of the operating system which is necessary for using resources of the computer.

WEEK 3

Questions

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?

This is a fork-bomb process : This process will create child process until memory is not full/ the system is not crashed.

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 real condition for exit the loop, a possibility to change the value in the while and if the process take too much time to change the loop condition, then stop him (runtime control).

c) would the controls that you propose hinder a process' ability to create new processes?

No, it is allow but they can't be create indefinitely, they are kill if they don't evolve fastly.

The problem is, that new processes create by the original process can also create processes : fix a limit of processes for a user can be a solution for limit the processes creation.

Virtual machine exercise:

16) Writing a short Shell script

Use `man` to find out what the `head` command does.

Using an editor (can be the simple editor you used in 1), 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.

It is a good idea to begin your script with a comment that explains what your script does. Comment lines begin with a hash (`#`). (You might have to experiment with the keyboard to find out which key produces the `#` on screen. For example, it might be the `£` key if you have got a British keyboard.)

Remember that you will have to make your script executable, using the `chmod` command (come out of the editor first).

Run your script by typing its name at the prompt. If it doesn't work, go back into the editor and modify it until it does.

```
head -n 5 test.txt | sort # we sort the lines of the text
chmod +x test #we the file with the good rights
```

WEEK 4

Questions

Tutorial questions:

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

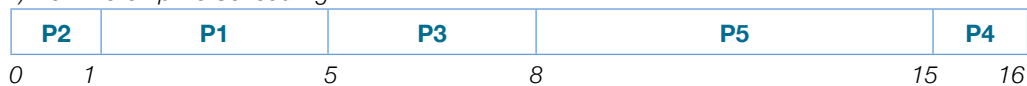
Process	Burst Time	Priority
P1	4	2
P2	1	1
P3	3	3
P4	1	4
P5	7	3

a) Draw four Gantt charts illustrating the execution of these processes using

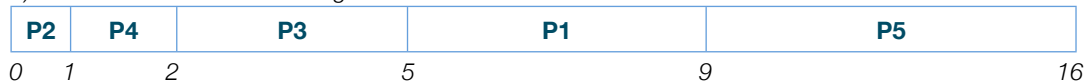
I) First Come First Serve Scheduling



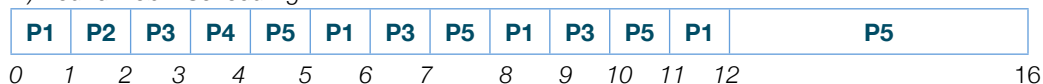
II) Non Pre-emptive Scheduling



III) Shortest-Job-First Scheduling



IV) Round-Robin Scheduling



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

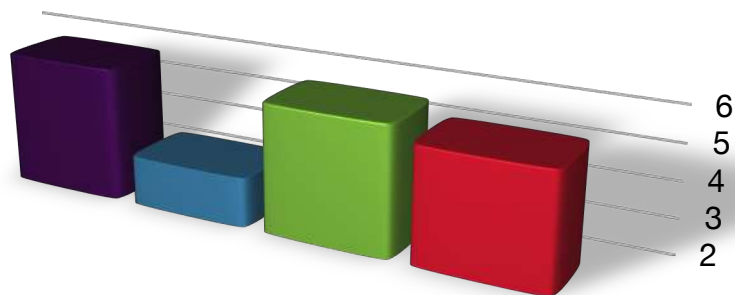
Type	P1	P2	P3	P4	P5
First Come First Serve	4	5	8	9	16
Shortest Job First	9	1	5	2	16
Non Pre-emptive	5	1	8	16	15
Round Robin	12	2	10	4	16

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

Type	P1	P2	P3	P4	P5
First Come First Serve	0	4	5	8	9
Shortest Job First	5	0	2	1	9
Non Pre-emptive	1	0	5	15	8
Round Robin	8	1	7	3	9

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

The Shortest Job First scheduling is the fastest schedule with a average waiting time of 3,4 :



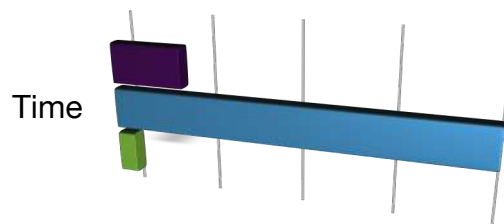
■ Non Pre-emptive Scheduling ■ Shortest Job First ■ Round-Robin ■ First Come First

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 .

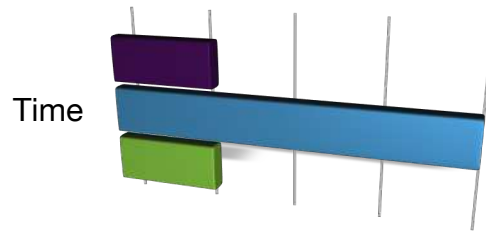
- q is slightly greater than zero overhead from context switch becomes bigger - when q becomes similar in length to context switch time then no work will get done - all context switching
- $q = s$
- $s < q < t$
- $q = t$
- $q > t$
- q is an extremely large number

■ Q ■ T ■ S

- For q very short, we need a s very very short or the program can't be launched.

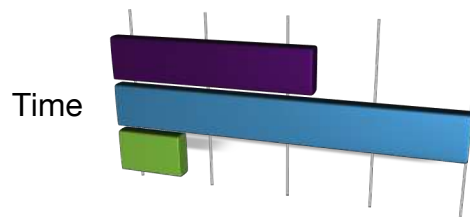


- b) we can load the process but useless because we just can load and unload the task but no time for execute

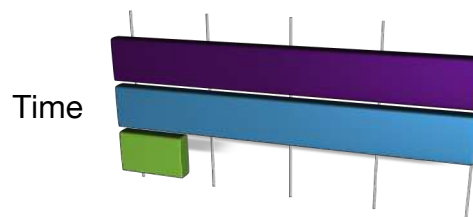


it!

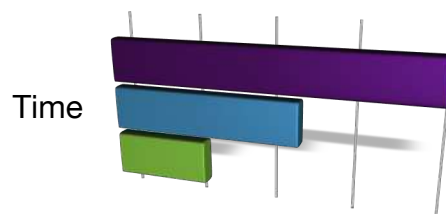
- c) This solution is possible but in round robin task witch divide our task very much.



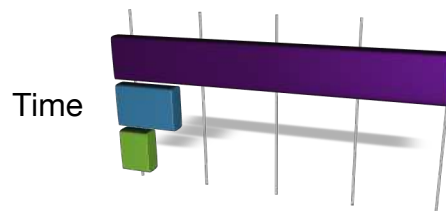
- d) This solution is possible in one time : this is the best solution, we don't lose so much time to wait and ask resources.



- e) We can load the program in one time but there is little waste time.



- f) This solution is possible and is close to FCFS solution

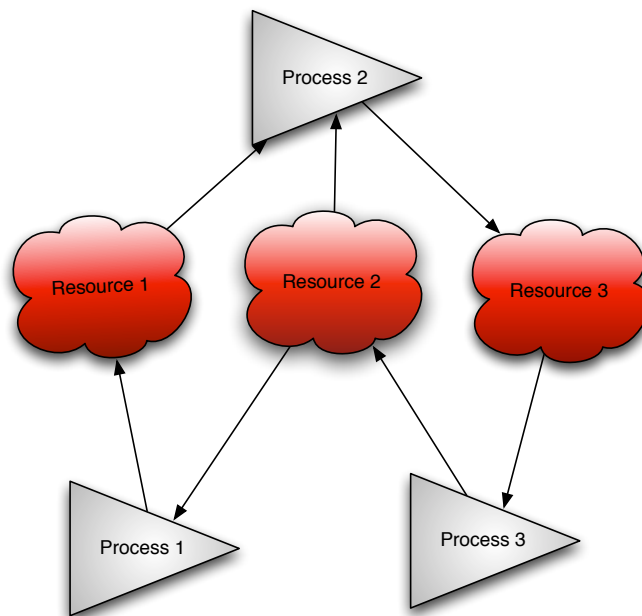


WEEK 5

Questions

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 an example of a circle deadlock :



On this example, process cannot access to their demand because :

Process 1 wait for Resource 1 use by Process 2.

Process 2 wait for Resource 3 use by Process 3.

Process 3 wait for Resource 2 instances use by Process 1 and Process 2.

Virtual machine exercises with Linux processes.**17)**

```

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

main()
{
    pid_t val;
    double j;
    int i;

    if (val=fork())
    {
        printf("PID after fork():%d - so who am I?\n", (int) getpid());

        for(i=1;i<=50;i=i+2)
        {
            printf("%d\t", i);
            for (j=0; j<=5000000; ++j);
        }
    }
    else
    {
        printf("PID after fork():%d - so who am I?\n", (int) getpid());
        for(i=2;i<=50;i=i+2)
        {
            printf("%d\t", i);
            for (j=0; j<=5000000; ++j);
        }
    }
}

```

23)

```

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

main()
{
    pid_t val;
    int status;

    printf("PID before fork(): %d\n", (int) getpid());
    if(val=fork())
    {
        printf("PID after fork(): %d - so who am I?\n\n", (int) getpid());
        wait(&status);
    }
    else
    {
        printf("PID after fork(): %d - so who am I?\n", (int) getpid());
        execl("/bin/ls", "ls", "-l", 0);
        status;
    }
}

```

WEEK 6

Questions

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.

The deny the «wait for» method is a technical process which can prevent deadlock on an operating system.

The action of this method is relatively easy to implement and really efficient but it is slow process because one application takes all resources and others wait their turn.

By this way, an application can create a starvation/use all the time resources and never share to others: it is an inefficient use of resources and creates difficulty to satisfy all processes.

Moreover, some applications release all their resources before requesting all the resources they will need which creates wasting of resources.

14) Virtual machine exercises

```
/* Basic cp file copy program. Win32 Implementation. */
/* cp file1: Copy file1 */
#include <windows.h>
#include <stdio.h>
#define BUF_SIZE 256
int main (int argc, LPTSTR argv [])
{
    HANDLE hIn, hOut;
    DWORD nIn, nOut;
    CHAR Buffer [BUF_SIZE];
    if (argc != 2) {
        printf ("Usage: cp file1\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;
    }
    /* We use GetStdHandle in hOut for print the file */
    hOut = GetStdHandle(STD_OUTPUT_HANDLE);
    if (hOut == GetStdHandle( STD_ERROR_HANDLE)) {
        printf ("Cannot printf file. Error: %x\n", GetLastError ());
        return 3;
    }
    while (ReadFile (hIn, Buffer, BUF_SIZE, &nIn, NULL) && nIn > 0) {
        WriteFile (hOut, Buffer, nIn, &nOut, NULL);
        if (nIn != nOut) {
            printf ("Fatal write error: %x\n", GetLastError ());
            return 4;
        }
    }
    CloseHandle (hIn);
    return 0;
}
```

WEEK 7

Questions

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?

Remember that space left over after a process has been placed in a partition will then become part of the free space available for later allocations.

First Fit

200K		450K		100K	700K		300K
112K	88K	212K	238K	100K	417K	283K	300K

426K WAITING

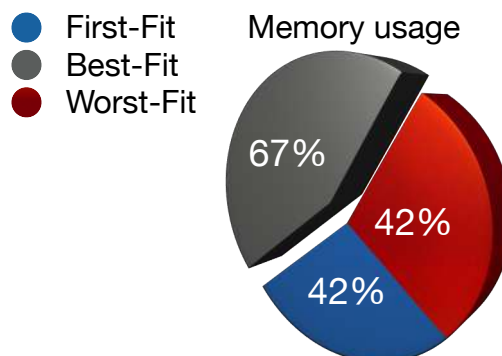
Best Fit

200K		450K		100K	700K		300K	
112K	88K	417K	33K	100K	426K	274K	212K	88K

Worst Fit

200K		450K		100K	700K		300K	
200K		112K	338K	100K	212K	417K	71K	300K

426K WAITING



The Best-Fit algorithm appears to be the best with 1167K used memory and no process waiting (versus 741K used and a 426K process in the waiting queue).

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 <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define SIZE 26

int main(int argc, char *argv[])
{
    int i, count, fd;
    char buf[SIZE];
    char ch;

    if((fd=open("shared", O_RDWR | O_CREAT, S_IRUSR | S_IWUSR | S_IRUSR | S_IWUSR)) == -1)
        printf("error opening file /n");

    if(fork()) {
        while (1){
            ch = 'a';
            for (i=0; i<SIZE; ++i)
                buf[i] = ch++;

            if(write(fd,buf,SIZE)!=SIZE)
                printf("error writing to file /n");
        }
    }
    else {
        {
            for(i=0;i<26;i++){
                printf("%c", buf[i]);
                if (read(fd,buf,SIZE)==-1) printf("error reading file/n");
            }
        }
        close(fd);
    }
}
```

WEEK 8

Questions

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? Remember all frames are initially empty, so your first unique pages will all cost one fault each.

- LRU replacement
- FIFO replacement
- Optimal replacement

FIFO 3 Frames															
1	1	1	1	4	4	4	4	4	3	3	3	3	1	1	1
	2	2	2	2	2	5	5	5	5	6	6	6	6	2	2
		3	3	3	3	3	2	2	2	2	2	5	5	5	3

With this algorithms there is 11 pages fault.

FIFO 4 Frames														
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	2	2	2	2	2
				4	4	4	4	4	4	4	4	1	1	1

With this algorithms there is 9 pages fault.

OPTIMAL 3 Frames														
1	1	1	1	4	4	5	5	5	5	5	5	1	1	3
	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

With this algorithms there is 8 pages fault.

OPTIMAL 4 Frames														
1	1	1	1	1	1	1	1	1	1	1	1	1	1	3
	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

With this algorithms there is 7 pages fault.

LRU 3 Frames													
1	1	1	1	4	4	4	4	3	3	3	5	5	3
	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

With this algorithms there is 10 pages fault.

LRU 4 Frames														
1	1	1	1	1	1	5	5	5	5	5	5	5	5	5
	2	2	2	2	2	2	2	2	2	2	2	2	2	2
		3	3	3	3	3	3	3	3	3	3	1	1	1
				4	4	4	4	4	6	6	6	6	6	3

With this algorithms there is 8 pages fault.

WEEK 9

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.

The largest file size can be calculate with : $1024 \text{ block} * 1024 \text{ block (2k blocks)} * 2 \text{ byte} = 2 \text{ Mbytes}$.

This result is very low limit actually, a image in JPEG with a correct definition is over 2 Mbytes!

With this kind of limit, computer are now useless, we count now in terabytes.

The limit fixed by NTFS v3.1 format file is 16 Tbytes, 8 000 000 more large than this system.

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

When you delete a file witch is used by other people, if you delete it, it's create a Invalid pointer to the file (obsolete) and create an error because user cannot access to this file: it is delete.

Delete a shared file make it unavailable for person who have access on it

10. How can we solve this problem?

For resolve that we have to wait that every user stop access to the file.

Making a delete list/count can be a solution:

Every time a user delete the file, the original count witch correspond of the number of the file users decrease until it superior at 0.

When every one has deleted it (count = 0), the file is really deleted.

WEEK 10

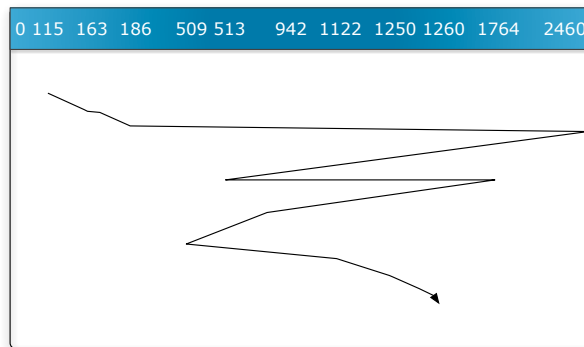
Questions

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

The First Come First Served method is scheduling method which follow the original sequence of requests.



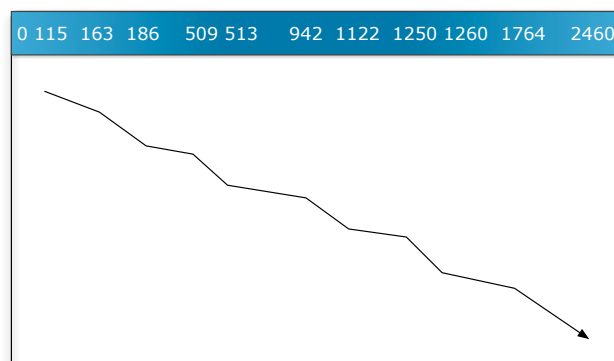
The graph of head movement cylinder in FCFS.

With this method, the total head movement is : $23 + 2274 + 1947 + 125 + 822 + 433 + 613 + 128 + 10 = \mathbf{7501}$

Also, there is 7501 movement of cylinder with FCFS sequence.

b. SCAN

The First Come First Served method is scheduling method choose to organise all the request into a continues way.



The graph of head movement cylinder in SCAN.

With this method, the total head movement is : $23 + 323 + 4 + 429 + 180 + 128 + 10 + 504 + 696 = \mathbf{2297}$

Also, there is 2297 movement of cylinder with SCAN sequence, 3x less than FCFS method!