## Problem 1

1. The part of the program that access a shared resources is considered as a critical region.
2. i) not taught anymore ii) monitor is a higher level of abstraction that semaphore, thus leave less room for mistakes iii) not taught anymore.

c)

Initialization:

Create semaphore size with initial value 100

Create semaphore lock with initial value 1

user\_process(Document d):

down(size)

down(lock)

q.join(d)

up(lock)

printer\_process1:

do forever

down(lock)

Document d = q.leave()

up(size)

up(lock)

print d on printer

end do

printer\_process2:

do forever

down(lock)

Document d = q.leave()

up(size)

up(lock)

print d on printer

end do

d)

It is possible to check the 4 conditions of the deadlock using the program, namely you can check for:

1. mutual exclusion
2. hold & wait
3. no preemption
4. circular wait

And if all 4 conditions are satisfied then the deadlock is possible.

## Problem 2

#include <stdio.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

int main(int argc, char \*argv[]) {

int status;

int pid = fork(); // fork a child process

if (pid == 0) { // code for child process

nice(10); // set the niceness of the child process to 10 so it will get executed after the main process

execlp("ls", "ls", NULL); // let the child process to execute the ls command, which list all the file in the current directory

}

printf("pid is %d.\n", pid); //

wait(&status);

return 0;

}

The outout is:

pid is 4469.

a.out exam.c

The assumption is that the main process has pid 4469, all the files that is in the current directory are a.out and exam.c

c)

short time quantum

mixed static and dynamic priority

high priority assigned to IO bound task

long time quantum

static priority

low priority for IO bound task

## Problem 3

a)

t\_access = t\_seek + t\_latency + t\_transfer

t\_seek = 5 + 100/10 = 15 ms

t\_latency = 1/(2\*100) = 0.005 s = 5 ms

t\_transfer = 1/100 \* 5/25 = 0.002s = 2 ms

t\_access = 15 + 5 + 2 = 22 ms

In the second case, we don’t have seek time any more since it is already in the same track.

So,

t\_access = 5 + 2 = 7ms

b)

SSTF 100 99 89 127 170 33 27

SCAN 100 127 170 99 89 33 27

C-SCAN 100 127 170 27 33 89 99

c) RAID uses multiple physical disks to store the information, however it is organized logically in the same way as one single virtual disk. the advantages of using RAID is that it greatly improves the disk capacity and it also introduces some error recovering and fault tolerant mechanism. Also it is cheaper to using RAID.

d) I don’t think we have covered this… Let me know if I am wrong.

e)

By increasing the sectors per track, assuming that the rotation speed keep unchanged, we know that the time it takes to transfer the data reduces. So the overall access time reduces.

If we increase the number of tracks per surface area that it means that the seek time increases is likely to increases. Thus the overall time for access is likely to increases.

## Problem 4

a)

See the lecture slide for a more detailed description

i) block linkage is an allocation method where for a given file, the previous block contains the pointer to the next block, the very last block’s pointer is NULL. The downside of the method is that it will need to always start from the first block which mean more block access.

ii) file allocation table contains a table which stores all the pointer to the relevant block.

iii) inode is a method that people stores an index block in which pointer to the block is stored. typically an inode consists of some direct pointer, some indirect pointer, some doubly indirect pointer and some triply indirect pointer.

b)

i)

the indirect pointer points to the bytes 0 - 6\* 1024, i.e. 0 - 6144 bytes.

the single-indirect pointer points to the bytes 6144 - 6144 + 256 \* 1024, i.e. 6144 - 268288

(18000 - 6144) / 1024 = 11 ... 592

(20000 - 6144) / 1024 = 13 ... 544

It will need 1 block read to obtain the block that contains the data block pointer

It will need 3 other block reads to obtained the 12th block, 13th block and 14th block to read the 18000 to 20000 bytes.

In total it will need 4 block reads.

ii) 6 + 1\*256 + 1\*256\*256 = 65798 blocks

c)

d)

i)

8\*4096 + 1\*(4096/8)\*4096 + 1\*(4096/8)\*(4096/8)\*4096 + 1\*(4096/8)\*(4096/8)\*(4096/8)\*4096 = 550831685632 bytes = 513 GB

ii)

2^4 = 16 times

rationale is that the triple-indirect pointer dominates so the total size will increase by roughly 16 fold (8200GB if we are using the data from i).