Homework 5

Problem 1

chmod (or “change mode”) dictates what the user/group that owns a file can do with that file.
chown (or “change owner”) dictates who owns a file.

(a)

chmod [character permission] [+/-/=] filename or path

sudo chown [maria] filename or path

setfacl -m u:maria:permissions

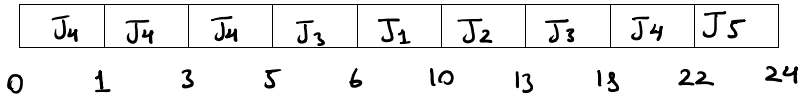
We use the “setfacl” which is a utility that can modify or remove ACL of files and directories.

Character	Explanation
u	User that owns the file
g	Group that owns the file
o	All other users and groups
a	All users and groups
r	Read permission
w	Write permission
x	Execute permission
–	Remove permission
+	Add permission
=	Make permissions exactly this

(b) Need-to-know principle decides about **what** and **which** permissions the user has while **least privilege** principle is about **how** the OS provide the permissions to the user.
According to least privilege principle, Maria has the privileges (chmod and chown permissions) in /home/sales while according to need-to-know principle she has no access to the rest of the system because she needs only to have rights only in her own division.

Problem 2

(a)



Job	Priority	AT	BT	CT	TAT	WT
J_1	1	6	4	10	4	0
J_2	2	6	3	13	7	4
J_3	3	5	6	18	13	7
J_4	4	1	7	22	21	14
J_5	5	3	2	24	21	19

Waiting time = turn-around time − burst time
Turn-around time = completion time − arrival time

(b)

Turnaround time is the interval from the time of submission of a process to the time of completion. It is the sum of the periods spent waiting in the ready queue, executing on the CPU and doing I/O.

It is the difference between completion time and burst time. Thus turnaround time for J1 is 0.

(c)

See the table in answer to the (a)

(d)

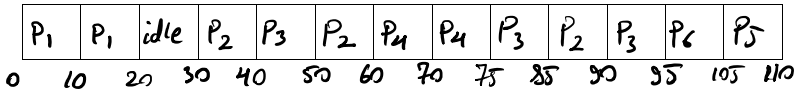
Even though J1 has the highest priority, but we consider that J4 and J3 processes has early arrival times than J1. If it was a non-preemptive SJF, then we could achieve the expectation in previous question.

Problem 3

(a)

Process	Priority	BT	AT	CT	TAT	WT
P_1	40	20	0	20	20	0
P_2	30	25	25	90	65	40
P_3	30	25	30	95	65	40
P_4	35	15	60	75	15	0
P_5	5	10	100	110	10	0
P6	10	10	105	105	0	—

TAT = CT − AT
WT = TAT − BT



(b)
According to the table in previous answer, waiting time for P3 is 30 and turnaround time for P3 is 55.

(c) Total burst time/total burst time + idle time **105/120 = 87.5 %**

Problem 4

If the context-switch time is approximately 10 percent of the time quantum, then about 10 percent of the CPU time will be spent in context switching.

(a) The CPU becomes idle during the **context-switch** and it depends on-scheduler how much time it take to assign the next process to CPU.

total burst time/ total burst time + idle time

1 / 1 + 0.2 = 0.83 => 83 %

(b) The time quantum should be large compared with the context switch time, meaning that the larger quantum times the lesser context switches and vice versa.

Problem 5

The head is positioned at 110 in the following requests:

75, 1471, 719, 1789, 1024, 1619, 1056, 1851, 120.

FCFS: 110, 75, 1471, 719, 1789, 1024, 1619, 1056, 1851, 120 = **7702**

SSTF: 110, 120, 75, 719, 1024, 1056, 1471, 1619, 1789, 1851 = **1831**

SCAN: 110, 75, 0, 120, 719, 1024, 1056, 1471, 1619, 1789, 1851 = **1961**

SSTF has much better performance than FCFS as its shown in results, but on the other hand it can suffer from starvation because it may not reach to the far requests as it serves the requests with the minimum seek time from the current head position.

Problem 6

(a) It may cause dynamic storage allocation problem which involves how to satisfy a request of size n from a list of free holes. This problem comes from external fragmentation where files are allocated and deleted, the free space is broken into little pieces. It becomes a problem when the largest contiguous piece is insufficient for a request.

(b) Linked list is more vulnerable because all other free blocks are pointed to the first free block which makes it easier to allocate to the other fast.

(c) It is inefficient because they are written to the device containing the file system for recovery needs). One way can be to keep them in main memory which is only possible for smaller devices.

Problem 7

Note: One-to-one model provides more concurrency than the many-to-one model. It also allows another thread to run when a thread makes a blocking system call. It supports multiple threads to execute in parallel on microprocessors.

Disadvantage of this model is that creating user thread requires the corresponding Kernel thread.

Note: All operations in the stack must be of O(1) time complexity

(a) Push ‘x’ to the stack. –> check if the last position in the stack –> delete the element from stack (if there is any element) and returns ‘x’

(b) Delete the element from stack on both user threads.

There will not be any problem as each user thread has its corresponding kernel thread.

Note: With kernel threads, the kernel is aware of all of the threads inside the process, because the kernel created them (on behalf of the application) and manages them directly, so the kernel can schedule any of them directly. Because of that, when thread A blocks inside a system call, the kernel/scheduler can go ahead and run thread B for a while instead, because the kernel knows that thread B exists. (Contrast this with the user-threads case, where the kernel can't schedule thread B to run, because the kernel doesn't know that thread B exists; only the user-application itself knows about the existence of the user-level threads)

Problem 8

(a) 0x113 —> 0x6B

(b) 0x113 —> 0x56

(c) 0xCF0 —> 0xA4

(d) 0xCF0 —> 0x21