Romil V. Shah
20008692

CS 520 – Intro to OS
Homework Assignment #3
Submission Date: Oct 11th, 2022

Note: Definitions have been taken as is from PDF material provided by the Professor and geeksforgeeks.com.

Problem 1.

Prove formally that the Shortest Job First scheduling algorithm is optimal in that it minimizes the average waiting time. For simplicity, assume that 1) all n processes are already in the system at the time the scheduling decision has to be made and 2) all processes have arrived at the same time. Hint: Use the formula for average waiting time in the Lecture.

Solution.

Shortest Job First is a scheduling policy that selects the waiting process with the smallest execution time to execute next. Shortest Job First is a non-preemptive algorithm. It may cause starvation if shorter processes keep coming. This problem can be solved using the concept of ageing.

The formula of the average waiting time is:

Average Waiting Time: $\frac{1}{n}\sum_{i=1}^{n-1}(n-i)t_i$

Let's assume there are set of n processes {P1, P2 … Pn} and their CPU burst time is {t1, t2, …,tn}. So, the average waiting time will be as shown below:

$W = ((n-1)\,t_1 + (n-2)\,t_2 + \ldots + (n-j)\,t_j + \ldots + t_{n-1}) / n$

From that n processes, taking two processes j and j-i. Their CPU burst time is t_j and t_{j-i} respectively where $t_{j-i} > t_j$ and $j > i$.

The average waiting time of process j will be

$W_j = ((n-1)\,t_1 + (n-2)\,t_2 + \ldots + (n-j)\,t_j + \ldots + (n-j+i)\,t_{j-i} + \ldots + t_{n-1}) / n$ ------ (1)

And for process j-i will be

$W_{j-i} = ((n-1)\,t_1 + (n-2)\,t_2 + \ldots + (n-j+i)\,t_j + \ldots + (n-j)\,t_{j-i} + \ldots + t_{n-1}) / n$ ------ (2)

Now subtracting equation (1) – (2),

$W_j - W_{j-i} = ((n-j+i)\,t_{j-i} + (n-j)\,tj - (n-j+i)\,t_j - (n-j)\,t_{j-i}) / n$

$= (n-j+I-n+j)\,t_{j-i} + (n-j-n+j-i)\,tj / n$

$= (i)\,t_{j-i} + (i)\,tj / n$

$W_j - W_{j-i} = i\,(t_{j-i} - t_j) / n$

Now, $t_{j-i} > t_j$ and $j > i$ which means the result will be greater than zero.

$W_j - W_{j-i} > 0$

$W_j > W_{j-i}$

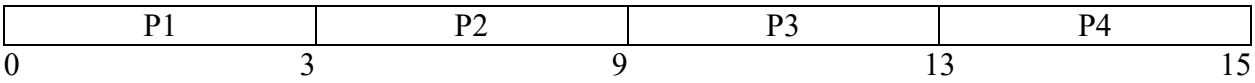
This means W_j will execute first than W_{j-i}. By repeating the execution of the shorter CPU burst time processes than the longer CPU burst time processes can result in minimal average waiting time. This way one can prove that the Shortest Job First scheduling algorithm is optimal in that it minimizes the average waiting time.

For example, there are 4 processes P1, P2, P3, and P4 and they all arrive at the same time 0. Their burst time is as follow:

Process	Burst time
P1	3
P2	6
P3	4
P4	2

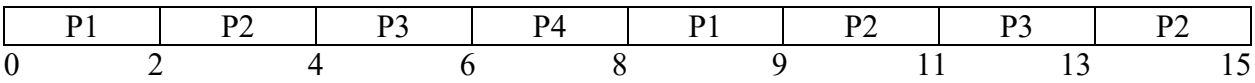
Gantt chart will be as shown below:

For First-Come-First-Serve algorithm:



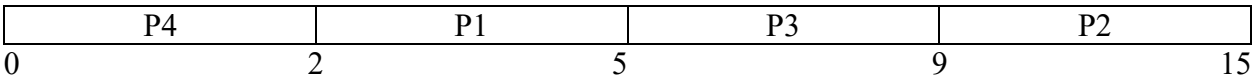
Average waiting time = (0 + 3 + 9 + 13) / 4 = 6.25 ms

For Round Robin algorithm (quantum = 2):



P1 = 8 – 2 = 6
P2 = 13 – 4 = 9
P3 = 11 – 2 = 9
P4 = 6 – 0 = 6
Average waiting time = (6 + 9 + 9 + 6) / 4 = 7.5 ms

For Shortest Job First algorithm:



Average Waiting Time = (0 + 2 + 5 + 9) / 4 = 4 ms

Here, Average Waiting Time for FCFS is 6.25 ms, for RR is 7.5 ms, and for SJF is 4 ms.

From the above example we can see that, Shortest Job First Scheduling is optimal as it has the lowest waiting time.

Problem 2.

The Snooty Clam restaurant does not take reservations. The dining room contains a single table seating twenty patrons. When space becomes free, parties are seated in the order in which they arrived, except that a party that cannot be seated in the available space is passed over. What is the effect of this seating policy on large parties? If parties are seated strictly in the order in which they arrive, how will this affect the utilization of the table?

Solution.

We have been given the situation as follows:
The Snooty Clam restaurant does not take reservations and they have a single table seating 20 patrons. When space becomes available, parties are seated as per their arrival. Except when the larger parties come first, and space is available for smaller parties then First Come First Served (FCFS) will not be applied and smaller parties will get seated.

Starvation is a problem that occurs when a process is waiting for a CPU (in a ready) state, but it never gets it, because new processes with higher priority always come in.

The problem of starvation can be resolved using aging. Aging, which increases the priority of a waiting process after a certain period it spent in a ready queue; thus, each process will eventually get a sufficiently high priority to run.

So here, after some time, space will be made available, but there is a great chance that the space that is made available is a single small party and thus the restaurant won't be able to accommodate larger parties who arrived first. Then again smaller parties will get seated first who arrived late. In this case, the wait time for large parties is considered to be more and might lead to indefinite wait time unless exact or a larger space is made available. This might result in the starvation for larger parties.

Let's consider again larger parties came first than the smaller parties. If parties are seated strictly in First Come First Serve (FCFS) order, then larger parties will seat first. In this case, smaller parties must wait until larger parties will be seated even if there is space available for smaller parties. This way starvation will not occur for larger parties and every party will get chance to seated. Only one drawback of this case is the average waiting time will increase because smaller parties cannot be seated even though space is available. This will affect in poor utilization of the table.

Let’s take an example for this,

The restaurant is currently empty. There are 13 different parties in a queue.
1. P1 – 3 People
2. P2 – 4 People
3. P3 – 18 People
4. P4 – 10 People
5. P5 – 4 People
6. P6 – 13 People
7. P7 – 5 People
8. P8 – 15 People
9. P9 – 3 People
10. P10 – 12 People
11. P11 – 7 People
12. P12 – 8 People
13. P13 – 2 People

Let’s say each party takes an average of 30 mins to eat and leave if it’s a small party and 40 mins if it’s a big party.

Note: Here, we take party of 10 or more to be a larger party. For simplicity, we also assume that all the parties arrive at the opening time.

As per the problem, we will see 2 aspects and see how much time it takes in both solutions.

Sol 1.

Table has 20 seats.
P1, P2, P4 and P9 gets seated. P3 is put on wait. (0 seats are empty.)

30 mins in, P1, P2 and P9 leave. (10 seats are empty now.)
P5 and P7 are seated. (Only 1 seat is empty.)

40 mins in P4 leaves. (11 seats are empty.)
P11 and P13 are seated. (2 seats are empty.)

60 mins in, P5 and P7 leave. (11 seats are empty.)
P12 is seated. (3 seats are empty.)

70 mins in, P11 and P13 leave. (12 seats are empty.)
P10 is seated. (0 seats are empty.)

90 mins in P12 leaves. (8 seats are empty.)

110 mins in, P10 leaves. (20 seats are empty.)
P3 is finally seated. (2 seats are empty.)

150 mins in, P3 leaves. (20 seats are empty.)
P6 is seated. (7 seats are empty.)

190 mins in, P6 leaves. (20 seats are empty.)
P8 is seated. (5 seats are empty.)

230 mins in, P8 leaves. (20 seats are empty.)

Here, we can see that the larger parties who have reserved have to wait if we let parties as per their size. This shows us that the space utilization done in this case is very good. This also shows us that the wait time of larger parties increase, and it might result in starvation.

If the restaurant has a closing time of 4 hours / 240 mins, then this solution will still be able to accommodate all the parties.

Sol 2. (Strictly FCFS):

Table has 20 seats.
P1 and P2 are seated. (13 seats are empty.)
30 mins in, P1 and P2 leave. (20 seats are empty.)

P3 is seated. (2 seats are empty.)
70 mins in, P3 leaves. (20 seats are empty.)

P4 and P5 are seated. (6 seats are empty.)
100 mins in, P5 leaves. (10 seats are empty.)
110 mins in, P4 leaves. (20 seats are empty.)

P6 and P7 are seated. (2 seats are empty.)
140 mins in, P7 leaves. (7 seats are empty.)
150 mins in. P6 leaves. (20 seats are empty.)

P8 and P9 are seated. (2 seats are empty.)
180 mins in, P9 leaves. (5 seats are empty.)
190 mins in, P8 leaves. (20 seats are empty.)

P10 and P11 are seated. (1 seat is empty.)
220 mins in, P11 leaves. (8 seats are empty.)

P12 is seated. (0 seats are empty.)
230 mins in, P10 leaves. (12 seats are empty.)

P13 is seated. (10 seats are empty.)
250 mins in, P12 leaves. (18 seats are empty.)
260 mins in, P13 leaves. (20 seats are empty.)

Here, we see that if the restaurant strictly follows First-Come-First-Served, then, all parties will be taken in as per their reservation/arrival no. and will be seated as per that. But we can also observe that the utilization of space in this solution is not optimal.

If the restaurant has a closing time of 4 hours / 240 mins, then this solution might not be able to accommodate all the parties.