Tony Maldonado
CS 474
November 9, 2020

# Homework #2

## Questions from Chapter 5

5.3 What is the meaning of the term busy waiting? What other kinds of waiting are there in an operating system? Can busy waiting be avoided altogether? Explain your answer.

- Busy waiting is essentially when a process is waiting for a condition to be satisfied in a tight loop without relinquishing the processor. There are other kinds of waiting; a process can wait by relinquishing or giving up the processor and block a condition, then wait to be awakened at some point in the future. Yes, busy waiting can be avoided but there is overhead that comes with it because of having to put the process to sleep and then having to wake up the process when it reaches the program state is reached.

5.10 Explain why implementing synchronization primitives by disabling interrupts is not appropriate in a single-processor system if the synchronization primitives are to be used in user-level programs?

- If the user-level programs are given the ability to disable interrupts, then the program can disable the timer interrupt and stop context switching from taking place which would then allow it to use the processor without allowing other processes to execute.

5.14 Describe how the compare and swap() instruction can be used to provide mutual exclusion that satisfies the bounded-waiting requirement.

```
do {
    waiting [ i ] = TRUE;
    key = TRUE;
    while (waiting [ i ] && key) {
            key = Swap(&lock, &key);
    }
    waiting [ i ] = FALSE;
    // Critical Section
    j = (i + 1) % n;
    while ((j != i) && !waiting[ j ]) {
            j = (j + 1) % n;
    }
    if (j == 1) {
            lock = FALSE;
    } else {
            waiting [ j ] = FALSE;
    }
    // remaining section
} while (TRUE);
```

Tony Maldonado
CS 474
November 9, 2020

5.17 Assume that a system has multiple processing cores. For each of the following scenarios, describe which is a better locking mechanism – a spinlock or a mutex lock where waiting processes sleep while waiting for the lock to become available:

- The lock is to be held for a short duration
- Because mutex requires suspending and then waking up a waiting process, a spinlock may be better.

- The lock is to be held for a long duration
- In this case I think a mutex lock would be better because this would allow another core to schedule other processes while the locked process waits for the lock to become available

- A thread may be put to sleep while holding the lock
- In this case a mutex lock would be better because you wouldn't want the process that is waiting to be spinning while waiting for another process to wake.

## Questions from Chapter 6

6.6 Suppose that a scheduling algorithm (at the level of short-term CPU scheduling) favors those processes that have used the least processor time in the recent past. Why will this algorithm favor I/O bound programs and yet not permanently starve CPU-bound programs?

- The algorithm would favor I/O bound programs because I/O bound programs have short CPU bursts, yet, the CPU bound programs won't starve because I/O bound programs will release the CPU somewhat often to do their I/O processes.

6.11 Discuss how the following pairs of scheduling criteria conflict in certain settings.

- CPU utilization and response time
- CPU utilization is increased if the overhead that comes with context switching is minimized, and the context switching overheads can be lowered by doing them less often. This could therefore result in increasing the response time for processes.

- Average turnaround time and maximize waiting time
- The average turnaround time is lowered by executing the shortest tasks first, but this could then starve long-running tasks and thereby increase their waiting time

- I/O device utilization and CPU utilization
- CPU utilization is maximized when running long CPU bound tasks without using context switching, where as I/O device utilization is maximized by

scheduling I/O bound tasks as soon as they're ready to run, which would incur the overhead of context switches.

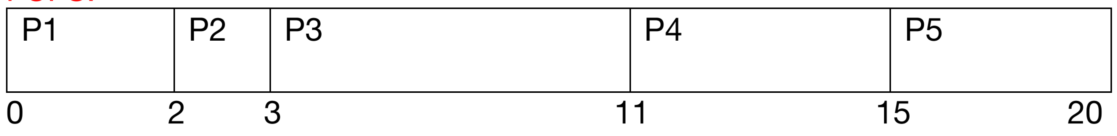
6.16 Consider the following set of processes, with the length of the CPU burst given in milliseconds:

| Thread | Burst | Priority |
|---|---|---|
| $P_1$ | 2 | 2 |
| $P_2$ | 1 | 1 |
| $P_3$ | 8 | 4 |
| $P_4$ | 4 | 2 |
| $P_5$ | 5 | 3 |

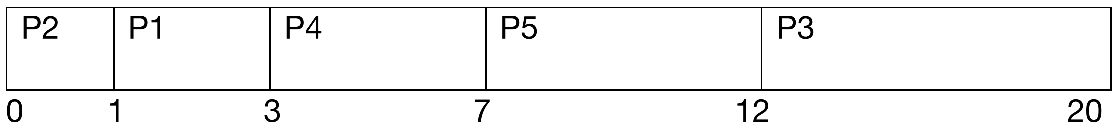
The processes are assumed to have arrived in the order $P_1$, $P_2$, $P_3$, $P_4$, $P_5$, all at time 0.

1. Draw four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF, non-preemptive priority (a larger priority number implies a higher priority), and RR (quantum = 2).

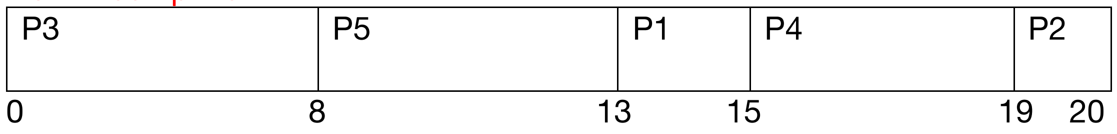
- FCFS:



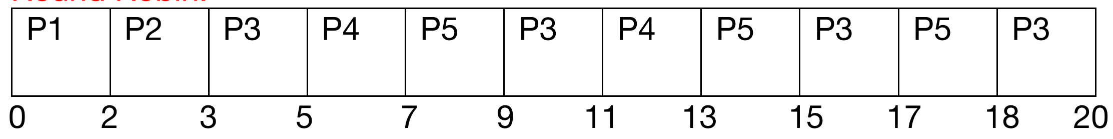
- SJF:



- Non-Preemptive:



- Round Robin:



2. What is the turnaround time of each process for each of the scheduling algorithms?

- FCFS:
P1: 2 - 0 = 2
P2: 3 - 0 = 3

P3: 11 – 0 = 11
P4: 15 - 0 = 15
P5: 20 - 0 = 20

- SJF:
  P1: 3 – 0 = 3
  P2: 1 – 0 = 1
  P3: 20 – 0 = 20
  P4: 7 – 0 = 7
  P5: 12 – 0 = 12

- Non-preemptive:
  P1: 15 - 0 = 15
  P2: 20 – 0 = 20
  P3: 8 – 0 = 8
  P4: 19 - 0 = 19
  P5: 13 -0 = 13

- Round Robin:
  P1: 2 – 0 = 2
  P2: 3 – 0 = 3
  P3: 8 – 0 = 8
  P4: 19 – 0 = 19
  P5: 13 – 0 = 13

3. What is the waiting time of each process for each of these scheduling algorithms?

- FCFS:
  P1: 2 – 2 = 0
  P2: 3 – 1 = 2
  P3: 11 - 8 = 3
  P4: 15 – 4 = 11
  P5: 20 – 5 = 15

- SJF:
  P1: 3 - 2 = 1
  P2: 1 – 1 = 0
  P3: 20 – 8 = 12
  P4: 7 – 4 = 3
  P5: 12 – 5 = 7

- Non-preemptive:

P1: 15 – 2 = 13
P2: 20 – 1 = 19
P3: 8 – 8 = 0
P4: 19 – 4 = 15
P5: 13 – 5 = 8

- Round Robin:
  P1: 2 - 2 = 0
  P2: 3 – 1 = 2
  P3: 20 – 8 = 12
  P4: 13 – 4 = 9
  P5: 18 - 5 = 13

4. Which of the algorithms results in the minimum average waiting time (over all processes)?

- FCFS Average = (0 + 2 + 3 + 11 + 15) / 5 = 31 / 5 = 6.2ms

- SJF Average = (1 + 0 + 12 + 3 + 7) / 5 = 23 / 5 = 4.6ms

- Non-preemptive Average = (13 + 19 + 0 + 15 + 8) / 5 = 55 / 5 = 11ms

- Round Robin Average = (0 + 2 + 12 + 9 + 13) / 5 = 36 / 5 = 7.2ms

- Therefore, SFJ has the minimum average waiting time.

Tony Maldonado
CS 474
November 9, 2020

6.17 The following processes are being scheduled using a preemptive, round-robin scheduling algorithm. Each process is assigned a numerical priority, with a higher number indicating a higher relative priority. In addition to the processes listed below, the system also has an idle task (which consumes no CPU resources and is identified as Pidle). This task has priority 0 and is scheduled whenever the system has no other available processes to run. The length of a time quantum is 10 units. If a process is preempted by a higher-priority process, the preempted process is placed at the end of the queue.

| Thread | Priority | Burst | Arrival |
|---|---|---|---|
| $P_1$ | 40 | 20 | 0 |
| $P_2$ | 30 | 25 | 25 |
| $P_3$ | 30 | 25 | 30 |
| $P_4$ | 35 | 15 | 60 |
| $P_5$ | 5 | 10 | 100 |
| $P_6$ | 10 | 10 | 105 |

1. Show the scheduling order of the processes using a Gantt chart

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P1 | P1 | Pidle | P2 | P3 | P2 | P3 | P4 | P4 | P2 | P3 | Pidle | P5 | P6 |

0 10 20 25 35 45 55 65 75 80 85 90 100 110 120

2. What is the turnaround time for each process?
   P1: 20
   P2: 85 – 25 = 60
   P3: 90 - 35 = 55
   P4: 80 – 65 = 15
   P5: 110 – 100 = 10
   P6: 120 – 110 = 10

3. What is the waiting time for each process?
   P1: 0
   P2: 10 + (80 – 55) = 35
   P3: 5 + 10 + (85 – 65) = 35
   P4: 5
   P5: 0
   P6: 5

4. What is the CPU utilization rate?
   Idle time = 15
   CPU utilizataion rate: (105 / 120) * 100 = 87.5%

Tony Maldonado
CS 474
November 9, 2020

## Questions from Chapter 7

7.10 Is it possible to have a deadlock involving only one single-threaded process? Explain your answer.

- No, it isn't. A deadlock involves 2 or more processes and you cannot have 'hold and wait' or 'circular wait' with only one single-threaded process.

7.16 In a real computer system, neither the resources available nor the demands of processes for resources are consistent over long periods (months). Resources break or are replaced, new processes come and go, and new resources are bought and added to the system. If deadlock is controlled by the banker's algorithm, which of the following changes can be made safely (without introducing the possibility of deadlock), and under what circumstances?

1. Increase Available (new resources added).
    - Can be changed safely and not cause any problems.

2. Decrease Available (resource permanently removed from system).
    - This could cause an issue that could lead to a deadlock even if it was in safe mode before because the system assumes there was a certain number of resources available before.

3. Increase Max for one process (the process needs or wants more resources than allowed).
    - This could have an effect on the system and cause a deadlock as new need might not be able to execute the program.

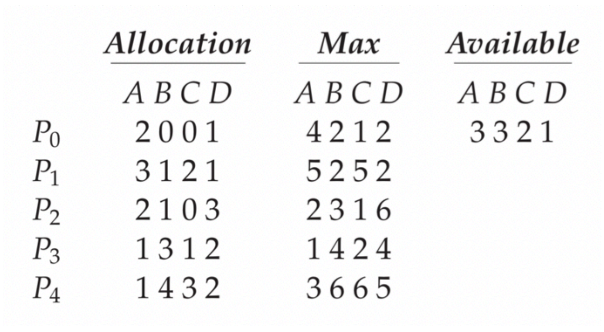
4. Decrease Max for one process (the process decides it does not need that many resources).
    - This couldn't cause deadlock, can be safely changed.

5. Increase the number of processes.
    - If the resources are allocated to the new processes and if the process don't enter an unsafe state, this can be done without causing deadlock.

6. Decrease the number of processes.
    - This could be done safely and not cause deadlock

Tony Maldonado
CS 474
November 9, 2020

7.23 Consider the following snapshot of a system:



| | *Allocation* | *Max* | *Available* |
|---|---|---|---|
| | *A B C D* | *A B C D* | *A B C D* |
| $P_0$ | 2 0 0 1 | 4 2 1 2 | 3 3 2 1 |
| $P_1$ | 3 1 2 1 | 5 2 5 2 | |
| $P_2$ | 2 1 0 3 | 2 3 1 6 | |
| $P_3$ | 1 3 1 2 | 1 4 2 4 | |
| $P_4$ | 1 4 3 2 | 3 6 6 5 | |

Answer the following questions using the banker's algorithm:

1. Illustrate that the system is in a safe state by demonstrating an order in which the processes may complete.

   - We need a matrix:

| | A | B | C | D |
|---|---|---|---|---|
| P0 | 2 | 2 | 1 | 1 |
| P1 | 2 | 1 | 3 | 1 |
| P2 | 0 | 2 | 1 | 3 |
| P3 | 0 | 1 | 1 | 2 |
| P4 | 2 | 2 | 3 | 3 |

   - Available: (3 3 2 1)
   - Need(P0) < Available so, P0 can take all resources
   - Available = (3 3 2 1) + (2 0 0 1) (Allocation of P0) = (5 3 2 2)
   - Need(P3) < Available so, P3 will go next
   - Available = (5 3 2 2) + (1 3 1 2) = (6 6 3 4)
   - Then next P4, P1, P2 will get resources too
   - So, the sequence in which the processes may complete is:
   - P0, P3, P4, P1, P2

2. If a request from process P1 arrives for (1, 1, 0, 0), can the request be granted immediately?
   - Since the request(P1) < available, we can grant the request.

Tony Maldonado
CS 474
November 9, 2020

3. If a request from process P4 arrives for (0, 0, 2, 0), can the request be granted immediately?

    - Since the request(P4) < available, we can grant the request.