Assignment 2 - Selected Solutions

5.5  A wait operation atomically decrements the value associated with a semaphore. If two wait operations are executed on a semaphore when its value is 1, if the two operations are not performed atomically, then it is possible that both operations might proceed to decrement the semaphore value thereby violating mutual exclusion.

5.8  This algorithm satisfies the three conditions of mutual exclusion.
1) Mutual exclusion is ensured through the use of the flag and turn variables. If both processes set their flag to true, only one will succeed, namely, the process whose turn it is. The waiting process can only enter its critical section when the other process updates the value of turn.

2) Progress is provided, again through the flag and turn variables. This algorithm does not provide strict alternation. Rather, if a process wishes to access their critical section, it can set their flag variable to true and enter their critical section. It sets turn to the value of the other process only upon exiting its critical section. If this process wishes to enter its critical section again—before the other process—it repeats the process of entering its critical section and setting turn to the other process upon exiting.

3) Bounded waiting is preserved through the use of the TTurn variable. Assume two processes wish to enter their respective critical sections. They both set their value of flag to true; however, only the thread whose turn it is can proceed; the other thread waits. If bounded waiting were not preserved, it would therefore be possible that the waiting process would have to wait indefinitely while the first process repeatedly entered—and exited—its critical section. However, Dekker’s algorithm has a process set the value of turn to the other process upon exiting, thereby ensuring that the other process will enter its critical section next.

6.11
a. CPU utilization and response time: CPU utilization is increased if the overheads associated with context switching is minimized. The context switching overheads could be lowered by performing context switches infrequently. This could, however, result in increasing the response time for processes.
b. Average turnaround time and maximum waiting time: Average turnaround time is minimized by executing the shortest tasks first. Such a scheduling policy could, however, starve long-running tasks and thereby increase their waiting time.
c. I/O device utilization and CPU utilization: CPU utilization is maximized by running long-running CPU-bound tasks without performing context switches. I/O device
utilization is maximized by scheduling I/O-bound jobs as soon as they become ready thereby incurring the overheads of context switches.

### 7.16

a. Increase Available (new resources added)—This could safely be changed without any problems.
b. Decrease Available (resource permanently removed from system)  
   —This could have an effect on the system and introduce the possibility of deadlock as the safety of the system assumed there were a certain number of available resources.
c. Increase Max for one process (the process needs more resources than allowed, it may want more)—This could have an effect on the system and introduce the possibility of deadlock.

d. Decrease Max for one process (the process decides it does not need that many resources)—This could safely be changed without any problems.
e. Increase the number of processes—This could be allowed assuming that resources were allocated to the new process(es) such that the system does not enter an unsafe state.
f. Decrease the number of processes—This could safely be changed without any problems.

### 7.19

The following rule prevents deadlock: when a philosopher makes a request for the first chopstick, do not grant the request if there is no other philosopher with two chopsticks and if there is only one chopstick remaining.

### 8.11

a. First-fit:
b. 115 KB is put in 300 KB partition, leaving (185 KB, 600 KB, 350 KB, 200 KB, 750 KB, 125 KB)
c. 500 KB is put in 600 KB partition, leaving (185 KB, 100 KB, 350 KB, 200 KB, 750 KB, 125 KB)
d. 358 KB is put in 750 KB partition, leaving (185 KB, 100 KB, 350 KB, 200 KB, 392 KB, 125 KB)
e. 200 KB is put in 350 KB partition, leaving (185 KB, 100 KB, 150 KB, 200 KB, 392 KB, 125 KB)
f. 375 KB is put in 392 KB partition, leaving (185 KB, 100 KB, 150 KB, 200 KB, 17 KB, 125 KB)
g. Best-fit:
h. 115 KB is put in 125 KB partition, leaving (300 KB, 600 KB, 350 KB, 200 KB, 750 KB, 10 KB)
i. 500 KB is put in 600 KB partition, leaving (300 KB, 100 KB, 350 KB, 200 KB, 750 KB, 10 KB)
j. 358 KB is put in 750 KB partition, leaving (300 KB, 100 KB, 350 KB, 200 KB, 392 KB, 10 KB)
k. 200 KB is put in 200 KB partition, leaving (300 KB, 100 KB, 350 KB, 0 KB, 392 KB, 10 KB)
l. 375 KB is put in 392 KB partition, leaving (300 KB, 100 KB, 350 KB, 0 KB, 17 KB, 10 KB)
m. Worst-fit:
n. 115 KB is put in 750 KB partition, leaving (300 KB, 600 KB, 350 KB, 200 KB, 635 KB, 125 KB)
o. 500 KB is put in 635 KB partition, leaving (300 KB, 600 KB, 350 KB, 200 KB, 135 KB, 125 KB)
p. 358 KB is put in 600 KB partition, leaving (300 KB, 242 KB, 350 KB, 200 KB, 135 KB, 125 KB)
q. 200 KB is put in 350 KB partition, leaving (300 KB, 242 KB, 150 KB, 200 KB, 135 KB, 125 KB)
r. 375 KB must wait

In this example, only worst-fit does not allow a request to be satisfied. An argument could be made that best-fit is most efficient as it leaves the largest holes after allocation. However, best-fit runs at time $O(n)$ and first-fit runs in constant time $O(1)$.

8.13

The contiguous memory allocation scheme suffers from external fragmentation as address spaces are allocated contiguously and holes develop as old processes die and new processes are initiated. It also does not allow processes to share code, since a process’s virtual memory segment is not broken into noncontiguous fine-grained segments. Pure segmentation also suffers from external fragmentation as a segment of a process is laid out contiguously in physical memory and fragmentation would occur as segments of dead processes are replaced by segments of new processes. Segmentation, however, enables processes to share code; for instance, two different processes could share a code segment but have distinct data segments. Pure paging does not suffer from external fragmentation, but instead suffers from internal fragmentation. Processes are allocated in page granularity and if a page is not completely utilized, it results in internal fragmentation and a corresponding wastage of space. Paging also enables processes to share code at the granularity of pages.

9.21  • 18

• 17

• 13