Process Management
Practice Exercises – Solutions
Winter 2025

Question 1: Fork-Exec (A)

6 times.

Question 2: Fork-Exec (B)

Only shareable memory pages (possibly code, possibly read-only static data) and content on the disk are shared. Copies of the stack, heap, and other non-shareable pages are made for the newly created process, and it gets its own registers, I/O buffers, etc.

Question 3: Multi-process Communication

Even on a single CPU core, it makes sense to use multiple threads if the I/O operations that need to be performed are blocking (thread cannot proceed until they complete). This way, another thread can use the CPU while those threads are waiting. If the operations are nonblocking, then not much will be gained from threads and the overhead of context switching would be worse than just using one thread.

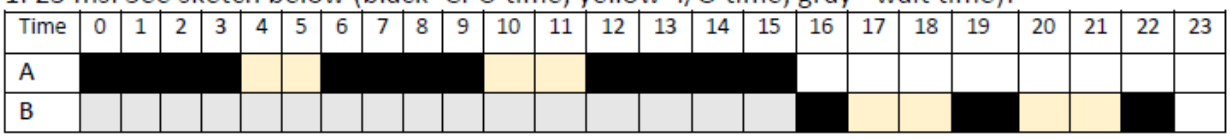
If the browser supports multiple tabs, then it is likely beneficial to use multiple *processes*, so that tabs can be made secure from each other via process separation. This would prevent a malicious website from reading your banking information, for example.

Question 4: Message Passing

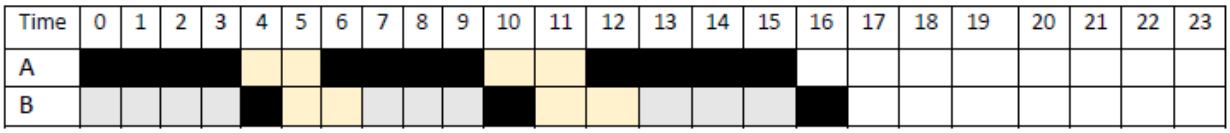
```c
struct MSG {
  char c;
  char str[1000]; // MUST be array, not pointer.
  int result;
  pid_t client_pid;
};

extern struct MSG *msg; // told to assume we have this

int client(char c, char *s) {
  msg->c = c;
  msg->client_pid = getpid();
  strcpy(msg->str, s);
  send_message(msg, SERVER_PID);
  recv_message(msg);
  return msg->result;
}

void server() {
  while (1) {
    recv_message(msg);
    msg->result = position(msg->c, msg->str);
    send_message(msg, msg->client_pid);
  }
}
```

Question 5: Scheduling (A)

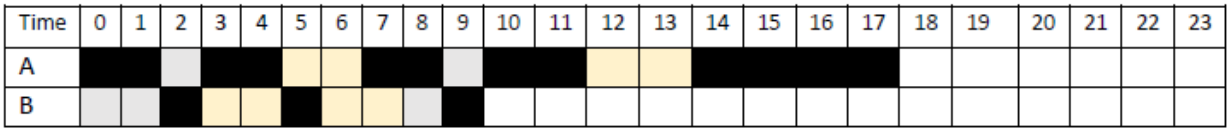
1. 23 ms. See sketch below (black=CPU time, yellow=I/O time, gray= wait time).



2. 17 ms. See sketch below.



3. 18 ms. See sketch below.



4. advantage:

- short job turnaround time is much better compared to non-preemptive (10ms vs 17ms)

Disadvantages:

- batch job (A) has longer turnaround time
- we assumed no context switching cost, but normally this is noticeable
- slightly longer execution time
- more idle CPU time in this example (because process B finishes too fast to “fill the gaps” later)

Question 6: Scheduling (B)

1. 10.53
2. 9.53
3. 6.86

Question 7: Semaphores

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 and the operations are not performed atomically, then both operations might decrement the semaphore value, thereby violating mutual exclusion.

Question 8: Dining Philosophers, Revisited

Many answers to this question are possible. To focus just on "explain why your solution avoids deadlocks," consider the solution where a semaphore is used so that one of the philosophers is not able to sit at the table at any given time.

In order for the dining philosophers to deadlock, a necessary condition is "circular waiting," where each philosopher is holding the chopstick on their left (for example) and waiting for the philosopher on their right to drop the chopstick between them. This solution avoids deadlocks by attacking this necessary condition. If one philosopher is not at the table, it is not possible for the philosopher to the left of the empty seat to be waiting for the chopstick between them, as that chopstick is necessarily available. Therefore deadlocks are impossible.