Process State Modeling

Operating Systems Lab Exercise 2

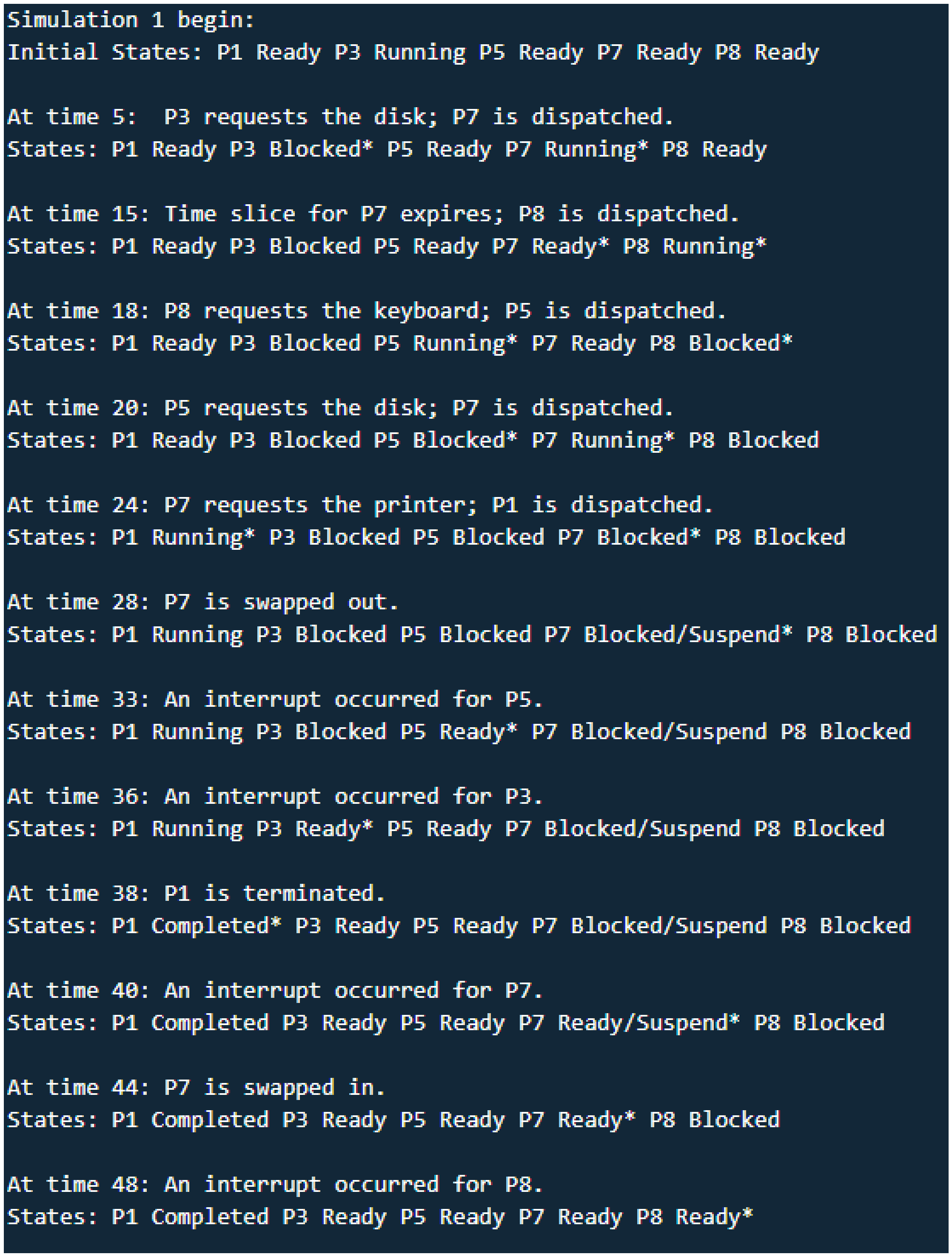
Skyelar Craver & Steven Pitts

February 9, 2019

[illegible] [illegible]

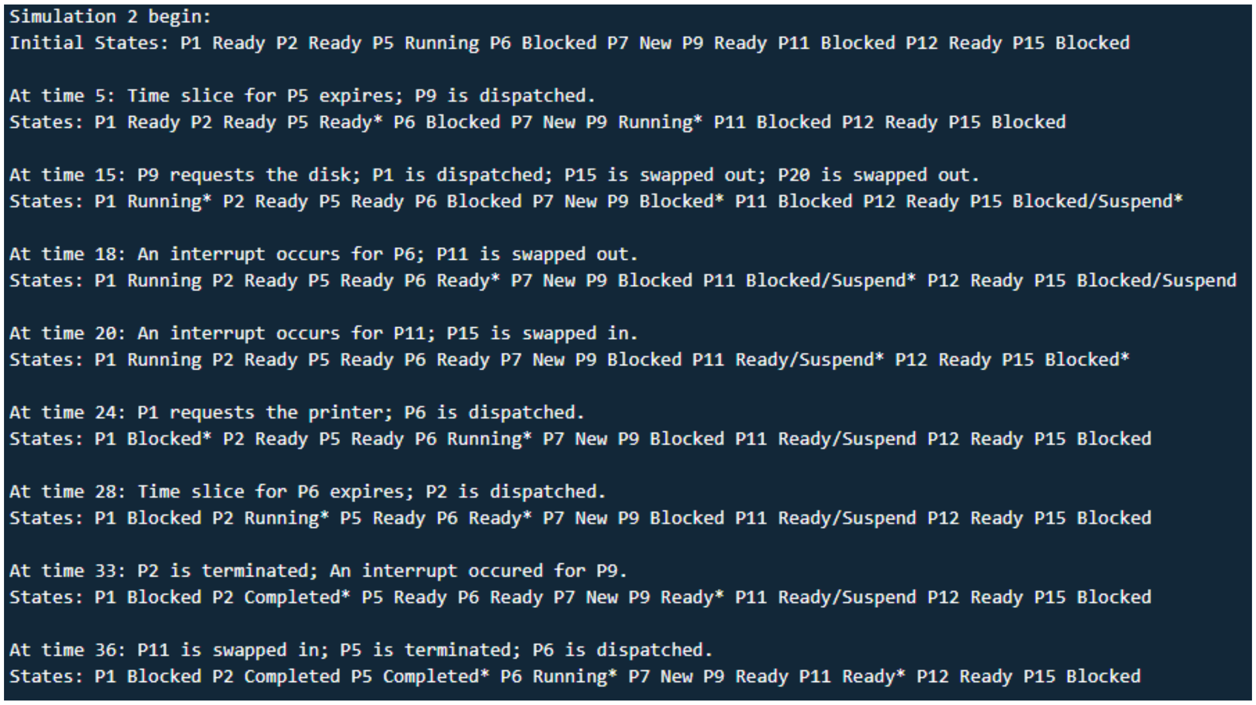
Part 1

Results:



```
Simulation 1 begin:
Initial States: P1 Ready P3 Running P5 Ready P7 Ready P8 Ready

At time 5:  P3 requests the disk; P7 is dispatched.
States: P1 Ready P3 Blocked* P5 Ready P7 Running* P8 Ready

At time 15: Time slice for P7 expires; P8 is dispatched.
States: P1 Ready P3 Blocked P5 Ready P7 Ready* P8 Running*

At time 18: P8 requests the keyboard; P5 is dispatched.
States: P1 Ready P3 Blocked P5 Running* P7 Ready P8 Blocked*

At time 20: P5 requests the disk; P7 is dispatched.
States: P1 Ready P3 Blocked P5 Blocked* P7 Running* P8 Blocked

At time 24: P7 requests the printer; P1 is dispatched.
States: P1 Running* P3 Blocked P5 Blocked P7 Blocked* P8 Blocked

At time 28: P7 is swapped out.
States: P1 Running P3 Blocked P5 Blocked P7 Blocked/Suspend* P8 Blocked

At time 33: An interrupt occurred for P5.
States: P1 Running P3 Blocked P5 Ready* P7 Blocked/Suspend P8 Blocked

At time 36: An interrupt occurred for P3.
States: P1 Running P3 Ready* P5 Ready P7 Blocked/Suspend P8 Blocked

At time 38: P1 is terminated.
States: P1 Completed* P3 Ready P5 Ready P7 Blocked/Suspend P8 Blocked

At time 40: An interrupt occurred for P7.
States: P1 Completed P3 Ready P5 Ready P7 Ready/Suspend* P8 Blocked

At time 44: P7 is swapped in.
States: P1 Completed P3 Ready P5 Ready P7 Ready* P8 Blocked

At time 48: An interrupt occurred for P8.
States: P1 Completed P3 Ready P5 Ready P7 Ready P8 Ready*
```

Figure 1 - output of simulation program for input file 1



```
Simulation 2 begin:
Initial States: P1 Ready P2 Ready P5 Running P6 Blocked P7 New P9 Ready P11 Blocked P12 Ready P15 Blocked

At time 5: Time slice for P5 expires; P9 is dispatched.
States: P1 Ready P2 Ready P5 Ready* P6 Blocked P7 New P9 Running* P11 Blocked P12 Ready P15 Blocked

At time 15: P9 requests the disk; P1 is dispatched; P15 is swapped out; P20 is swapped out.
States: P1 Running* P2 Ready P5 Ready P6 Blocked P7 New P9 Blocked* P11 Blocked P12 Ready P15 Blocked/Suspend*

At time 18: An interrupt occurs for P6; P11 is swapped out.
States: P1 Running P2 Ready P5 Ready P6 Ready* P7 New P9 Blocked P11 Blocked/Suspend* P12 Ready P15 Blocked/Suspend

At time 20: An interrupt occurs for P11; P15 is swapped in.
States: P1 Running P2 Ready P5 Ready P6 Ready P7 New P9 Blocked P11 Ready/Suspend* P12 Ready P15 Blocked*

At time 24: P1 requests the printer; P6 is dispatched.
States: P1 Blocked* P2 Ready P5 Ready P6 Running* P7 New P9 Blocked P11 Ready/Suspend P12 Ready P15 Blocked

At time 28: Time slice for P6 expires; P2 is dispatched.
States: P1 Blocked P2 Running* P5 Ready P6 Ready* P7 New P9 Blocked P11 Ready/Suspend P12 Ready P15 Blocked

At time 33: P2 is terminated; An interrupt occured for P9.
States: P1 Blocked P2 Completed* P5 Ready P6 Ready P7 New P9 Ready* P11 Ready/Suspend P12 Ready P15 Blocked

At time 36: P11 is swapped in; P5 is terminated; P6 is dispatched.
States: P1 Blocked P2 Completed P5 Completed* P6 Running* P7 New P9 Ready P11 Ready* P12 Ready P15 Blocked
```

Figure 2 - simulation program output using input file 2

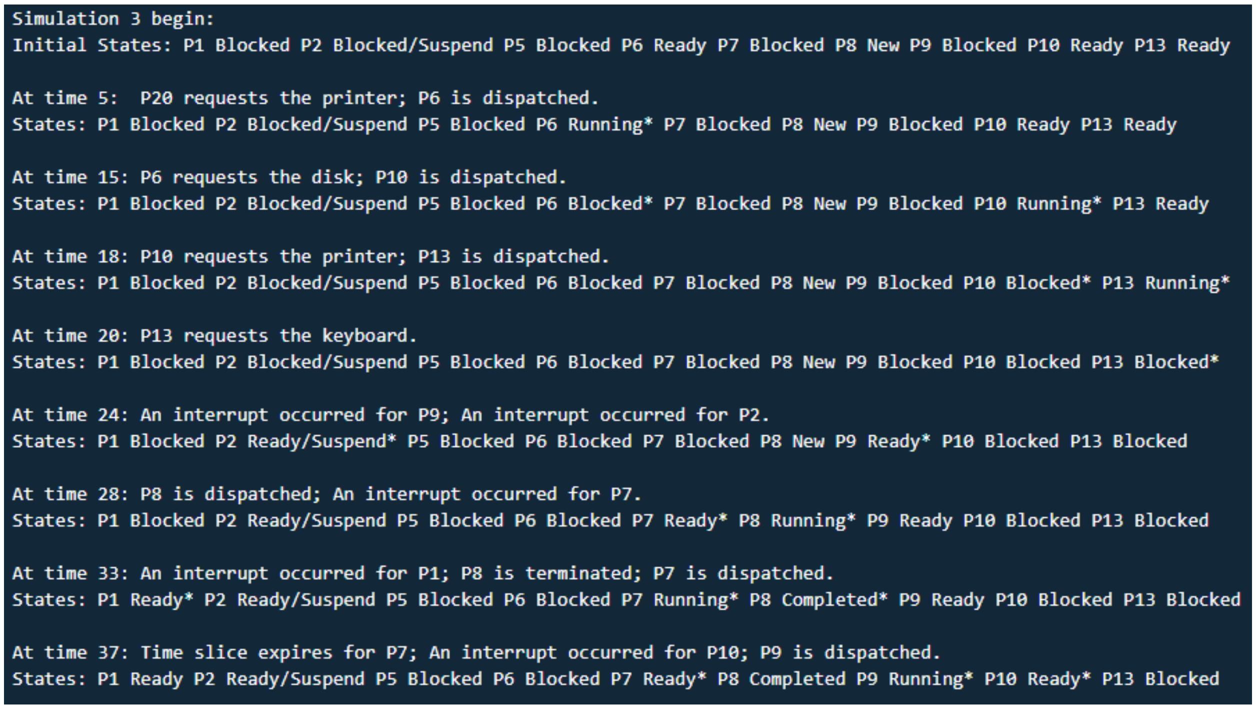
Code:

```
    prepare_variables();
    FILE *fp = fopen(file_paths[file_num], "r");
    fgets(current_line, 500, fp);
    process_inital_states(current_line);
    while (fgets(current_line, 500, fp) != NULL)
    {
        int states_changed[20] = {0};
        printf("\n%s", current_line);
        int curr_time = get_first_int(current_line);
        line_index = 0;
        while (!reached_end)
        {
            char *sub_line = get_next_command(current_line);
            action next_action = get_action(sub_line);
            int current_process = get_pid(sub_line);
            states_changed[current_process] = 1;
            processes[current_process] = switch_on_process_action(sub_line);
        }
        print_states(states_changed);
    }
    fclose(fp);
```

Code block 1 – pseudo code of main function for files 1 and 2.

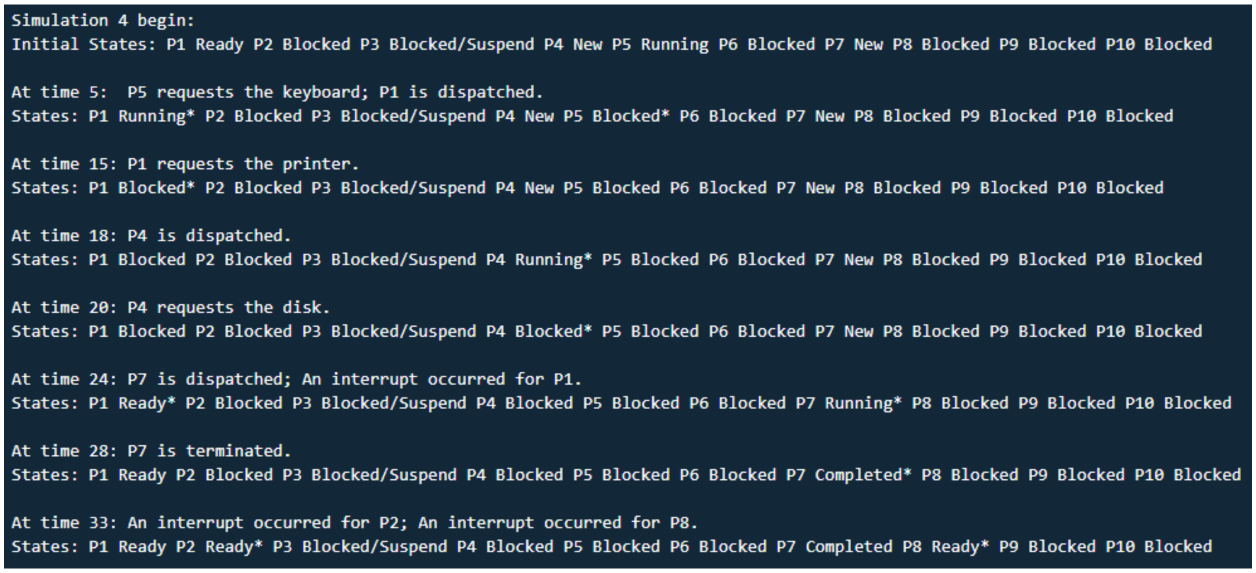
Part 2

Results:



```
Simulation 3 begin:
Initial States: P1 Blocked P2 Blocked/Suspend P5 Blocked P6 Ready P7 Blocked P8 New P9 Blocked P10 Ready P13 Ready

At time 5:  P20 requests the printer; P6 is dispatched.
States: P1 Blocked P2 Blocked/Suspend P5 Blocked P6 Running* P7 Blocked P8 New P9 Blocked P10 Ready P13 Ready

At time 15: P6 requests the disk; P10 is dispatched.
States: P1 Blocked P2 Blocked/Suspend P5 Blocked P6 Blocked* P7 Blocked P8 New P9 Blocked P10 Running* P13 Ready

At time 18: P10 requests the printer; P13 is dispatched.
States: P1 Blocked P2 Blocked/Suspend P5 Blocked P6 Blocked P7 Blocked P8 New P9 Blocked P10 Blocked* P13 Running*

At time 20: P13 requests the keyboard.
States: P1 Blocked P2 Blocked/Suspend P5 Blocked P6 Blocked P7 Blocked P8 New P9 Blocked P10 Blocked P13 Blocked*

At time 24: An interrupt occurred for P9; An interrupt occurred for P2.
States: P1 Blocked P2 Ready/Suspend* P5 Blocked P6 Blocked P7 Blocked P8 New P9 Ready* P10 Blocked P13 Blocked

At time 28: P8 is dispatched; An interrupt occurred for P7.
States: P1 Blocked P2 Ready/Suspend P5 Blocked P6 Blocked P7 Ready* P8 Running* P9 Ready P10 Blocked P13 Blocked

At time 33: An interrupt occurred for P1; P8 is terminated; P7 is dispatched.
States: P1 Ready* P2 Ready/Suspend P5 Blocked P6 Blocked P7 Running* P8 Completed* P9 Ready P10 Blocked P13 Blocked

At time 37: Time slice expires for P7; An interrupt occurred for P10; P9 is dispatched.
States: P1 Ready P2 Ready/Suspend P5 Blocked P6 Blocked P7 Ready* P8 Completed P9 Running* P10 Ready* P13 Blocked
```

Figure 3 - simulation program output using input file 3



```
Simulation 4 begin:
Initial States: P1 Ready P2 Blocked P3 Blocked/Suspend P4 New P5 Running P6 Blocked P7 New P8 Blocked P9 Blocked P10 Blocked

At time 5:  P5 requests the keyboard; P1 is dispatched.
States: P1 Running* P2 Blocked P3 Blocked/Suspend P4 New P5 Blocked* P6 Blocked P7 New P8 Blocked P9 Blocked P10 Blocked

At time 15: P1 requests the printer.
States: P1 Blocked* P2 Blocked P3 Blocked/Suspend P4 New P5 Blocked P6 Blocked P7 New P8 Blocked P9 Blocked P10 Blocked

At time 18: P4 is dispatched.
States: P1 Blocked P2 Blocked P3 Blocked/Suspend P4 Running* P5 Blocked P6 Blocked P7 New P8 Blocked P9 Blocked P10 Blocked

At time 20: P4 requests the disk.
States: P1 Blocked P2 Blocked P3 Blocked/Suspend P4 Blocked* P5 Blocked P6 Blocked P7 New P8 Blocked P9 Blocked P10 Blocked

At time 24: P7 is dispatched; An interrupt occurred for P1.
States: P1 Ready* P2 Blocked P3 Blocked/Suspend P4 Blocked P5 Blocked P6 Blocked P7 Running* P8 Blocked P9 Blocked P10 Blocked

At time 28: P7 is terminated.
States: P1 Ready P2 Blocked P3 Blocked/Suspend P4 Blocked P5 Blocked P6 Blocked P7 Completed* P8 Blocked P9 Blocked P10 Blocked

At time 33: An interrupt occurred for P2; An interrupt occurred for P8.
States: P1 Ready P2 Ready* P3 Blocked/Suspend P4 Blocked P5 Blocked P6 Blocked P7 Completed P8 Ready* P9 Blocked P10 Blocked
```

Figure 4 - simulation program output using input file 4

Code:

```
prepare_variables();
    FILE *fp = fopen(file_paths[file_num], "r");
    fgets(current_line, 500, fp);
    process_inital_states(current_line);
    while (fgets(current_line, 500, fp) != NULL)
    {
        int states_changed[20] = {0};
        printf("\n%s", current_line);
        int curr_time = get_first_int(current_line);
        line_index = 0;
        while (!reached_end)
        {
            char *sub_line = get_next_command(current_line);
            action next_action = get_action(sub_line);
            int current_process = get_pid(sub_line);
            states_changed[current_process] = 1;
            processes[current_process] = switch_on_process_action(sub_line);
        }
        if (check_need_to_swap())
        {
            suspend_blocked_process();
            prepare_new_or_suspended_process();
        }
        print_states(states_changed);
    }
    fclose(fp);
    return 1;
```

code block 2 – pseudo code of main function for part 2 with added parts highlighted.

Questions:

- When all of the Processes are Blocked, how did you choose which Process to swap out, and why did you choose that method?
    - If all processes are blocked, the lowest number process is the process swapped.
- List any other alternative methods for choosing which to swap out.
    - Keeping track of how long processes have been blocked and swapping based on that, or random selection among the blocked processes could be used.
- In a scenario where there are several Processes in the Ready/Suspend state, when a Process exits, how did you choose which Ready/Suspend Process to swap back in from the disk, and why did you choose that method?
    - The process list is scanned from lowest process number when looking for a new process to run. This approach was easiest to implement yet is biased and may cause some processes to be suspended for a very long time.
- In a scenario where there are new Processes and Ready/Suspend Processes available, to enter memory, which would you choose to enter memory? Give an example of a possible case where it is better to bring in a new Process. Give an example of a possible case where it is better to bring in a Ready/Suspend Process.
    - During the same scan that's performed to swap in suspended processes, a new process may be added. Whichever is found first, new or suspended, will be swapped in.
- In the input files provided you have Ready/Suspend Processes available to be swapped in, or a new Process waiting. Discuss your approach to the following scenario –a Process exits therefore there is space free in memory, however, all the Processes on the disk are still in the Blocked/Suspend state and there are no new Processes waiting. How would you handle this: do you leave all Processes on the disk, do you still swap one in, some other alternative? Explain why you chose your method.
    - Our approach does nothing in this scenario. Our algorithm will try to find a new or ready suspended process to swap in every cycle, but if one is not found, then no action is taken. An additional algorithm could be implemented which kept track of how long each process has been blocked, and swap in the process that has been blocked the longest – estimating that that process will be the most likely to be ready. Our lazy approach does save the time needed for IO however, perhaps meaning that when a process becomes unblocked it can be addresses more quickly.