M1S2P – Jacob Booth 221530207

Part 1

1. Identify what is stored in the text segment when the process is admitted by the OS and is in the ready state.
The text segment stores the program code functions in the form of compiled machine code. This includes the written functions in the program sum() and main(), and functions included in standard library code like the references to cout/cin. The OS admits the process, and when it's in a ready state the function machine code sits in the text segment of memory. **[1]**

2. Identify what variables will be stored in the data segment.
The data segment for a process holds the global and static variables which have been initialised explicitly, which would be int size = 4 (global) and static int result = 0 (local static).

3. Identify what is stored in heap and stack segments when line 12 is being executed for the third time.
On the third iteration of the main loop, i == 2:

Stack:
Stack memory holds local variables and the function call stack of the currently-scoped function(s). The function call stack contains things like the local function variables, arguments, return address, etc.

- Local vars: p pointer, i == 2, int total == p[0] + p[1] + p[2] (on iteration complete, so 4 + 5 + 8 = 17)
- Function stack frames: main() and sum()

Heap:
Heap memory holds the array being summed, integers 4, 5, and 8 have been stored here.

- Dynamically allocated int array: p = new int[size] where p[0], p[1], p[2] are filled with 4, 5, and 8.

4. Identify what is stored in heap and stack segments when line 33 is being executed.
When line 33 is executed, the loop has finished all iterations:

Stack:
- Local vars: index i = 4, and total = sum(p[0], p[1], p[2], p[3]), so the sum 4.0 + 5 + 8 + 3 = 20
- Function stack frames: only main() with its local vars since this step lies outside sum() execution scope

Heap:
After calling delete p, the memory that was assigned to the array has been cleared, so the array is removed from heap and only the pointer remains in the stack frame of main()

Part 2

1. Based on the design and the code, explain what the primary function of this board is. Complete the code by adding appropriate comments in the designated lines.

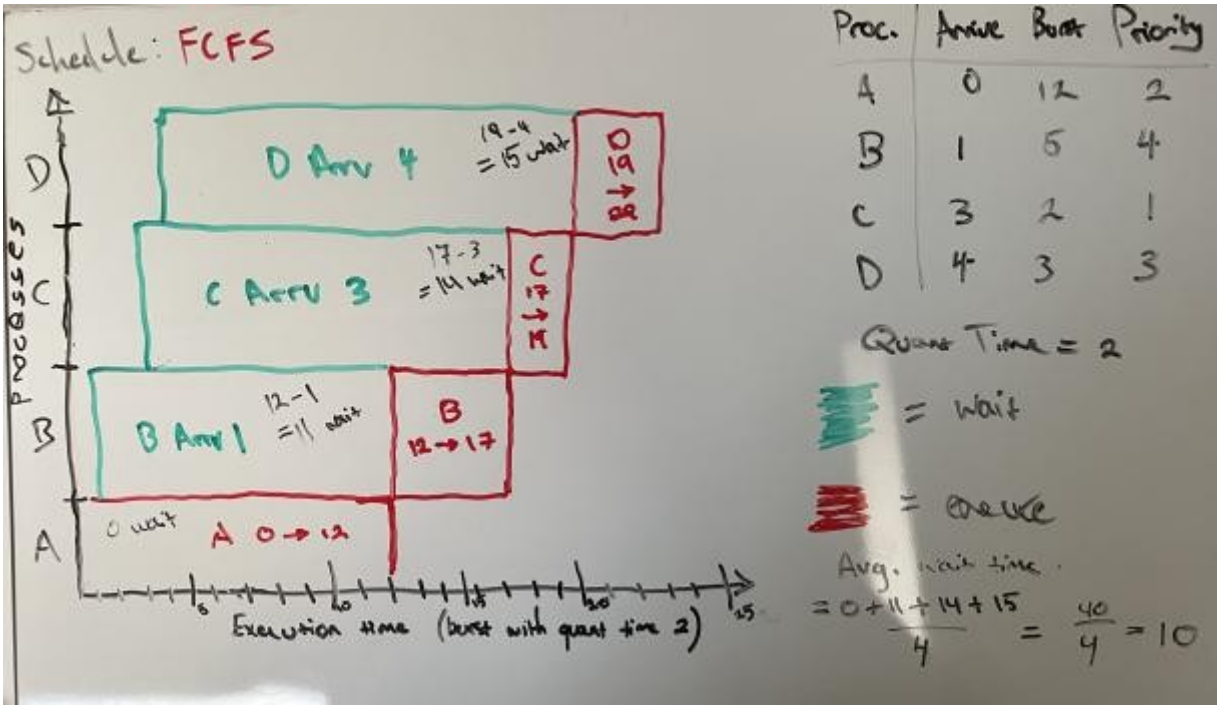
The code is designed to toggle an LED on and off when the user presses it, but the behaviour corresponds to the light being toggled only when the user holds their finger down on the button, no toggle. There's also some pretty noticeable latency, so the sleep(500) creates a janky experience. I've added comments to address each stage of the script.

2. Identify what the main problem in the code is and how it can affect the end-users.
An LED that's only lit when a button is held doesn't provide a whole lot of benefit, so I've assumed that this is meant to be a toggle LED for each press. In the code, LED state is toggled for every time the button state is changed. The delay(500) also introduces a lot of latency. This can be subbed for a much smaller bounce delay to ensure distinction between UP/DOWN states without introducing so much latency. The loop logic needs to be reconsidered to make sure the LED is only toggled when pressed down, ignoring the rise.

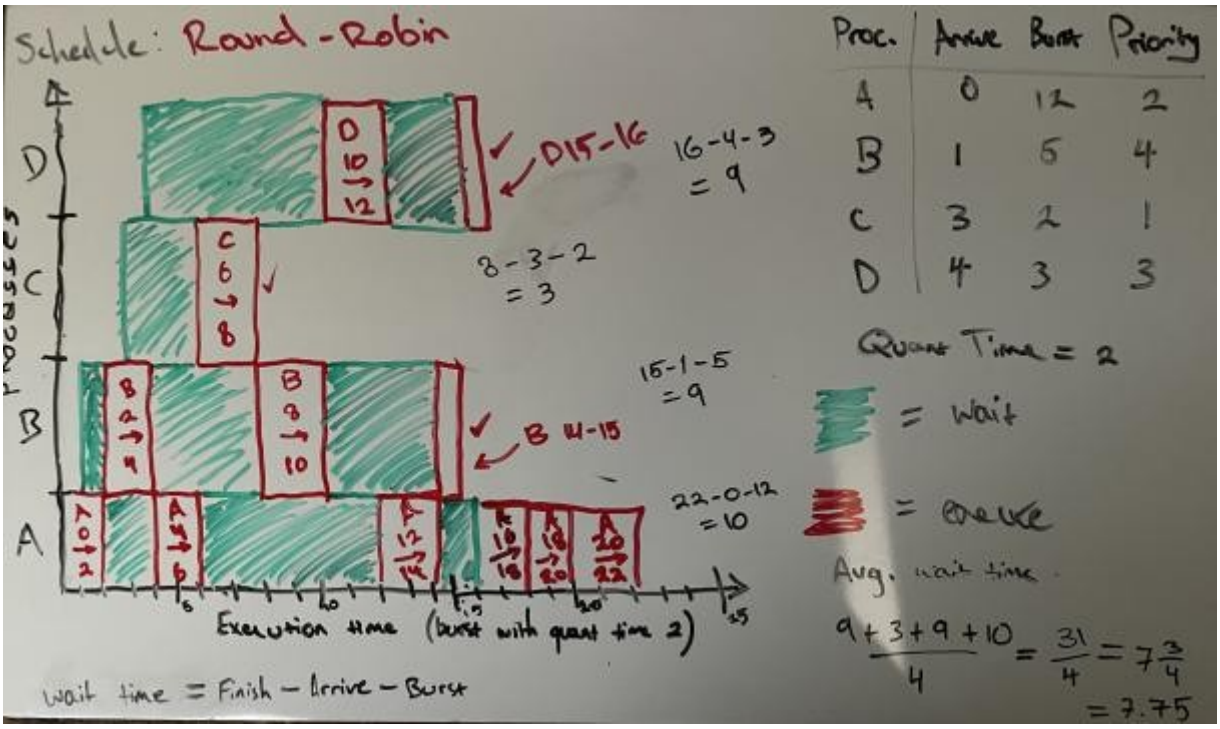
3. Change the code to resolve the problem you identified in 2.
I altered the conditional loop to only alter the state on button down and not on button up, where the down-click triggers the LED state toggle only if the previous button state was not pushed. I also made the delay much shorter and moved it inside the loop, making the responsiveness much snappier.

Part 3

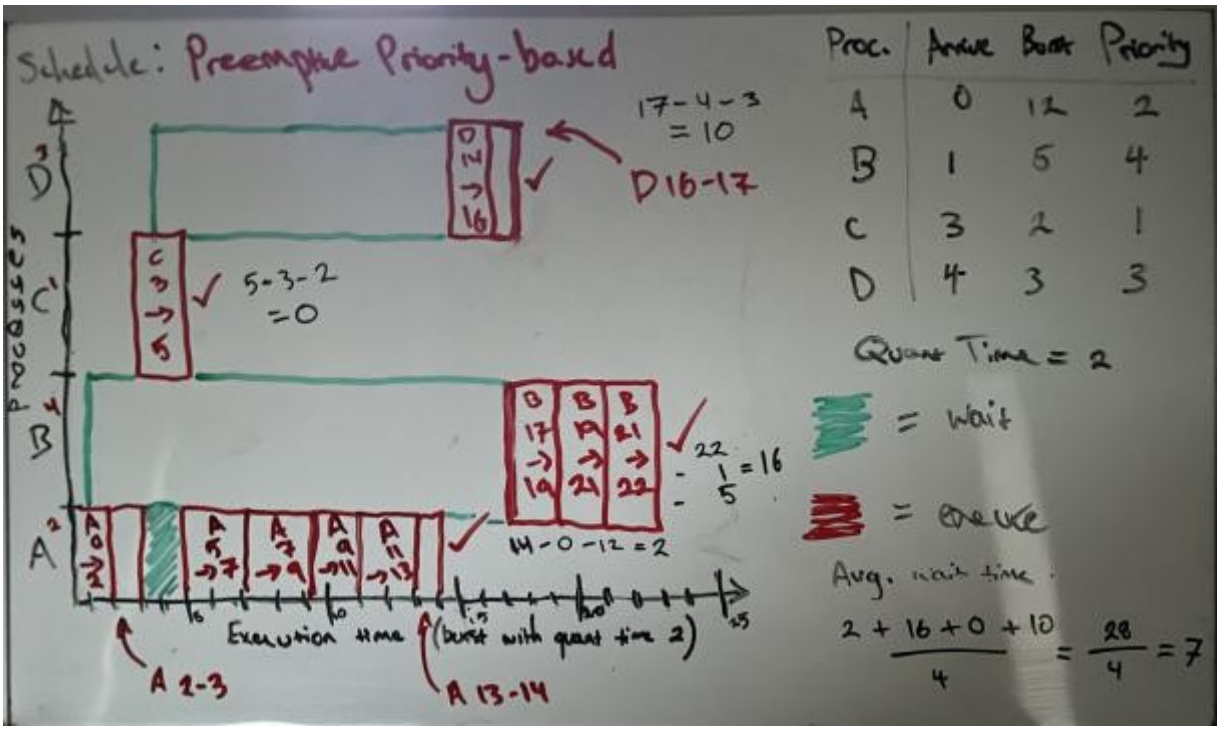
1. Draw three Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, Round-Robin, and Preemptive Priority-based scheduling.
2. For each of the scheduling algorithms, compute the waiting times of each process.
3. Compute the average waiting time of each scheduling algorithm.



[2]



[3]



NOTE: I used a bottom x-axis for these diagrams to emphasise scheduling distribution of the earliest processes (bottom) and compare it with the scheduling complexity as the schedule grows to accommodate A – D, though Gantt charts usually use a top x-axis.

4. Research and find another scheduling algorithm that has a lower waiting time than these three algorithms

Shortest Remaining Time First (SRTF)

SRTF prioritises workload over arrival time/priority. The scheduler compares remaining burst times for queued processes and preempts when a shorter task arrives. If the new process has lower remaining burst time than the one currently executing, it will be preempted and bumped to execution at the next quantum. This takes a preemptive approach to SJF, so queued processes with lower remaining burst time don't need to wait for the in-progress process to finish executing. This results in shorter overall waiting time than the previous 3 schedulers, at the risk of significantly longer tasks being continuously delayed due to shorter incoming processes. The context switch overhead is also increased due to the increased switching required in scenarios with lots of incoming, short tasks. This results in SRTF juggling between them frequently, requiring regular state-switches. It also relies on the burst time being known, or at least predictable, for each process. The more unpredictability that's introduced to the process queue, the less practical it will be. **[4]**

References

[1] M. Abbas, "Stack vs Heap: What's the difference? " Educative. [Online].
Available: https://www.educative.io/blog/stack-vs-heap.
[Accessed: 19-Jul-2025].

[2] GeeksforGeeks, "Round Robin Scheduling in Operating System," GeeksforGeeks. [Online].
Available: https://www.geeksforgeeks.org/operating-systems/round-robin-scheduling-in-operating-system/.
[Accessed: 19-Jul-2025].

[3] GeeksforGeeks, "Preemptive Priority CPU Scheduling Algorithm," GeeksforGeeks. [Online].
Available: https://www.geeksforgeeks.org/operating-systems/preemptive-priority-cpu-scheduling-algortithm/.
[Accessed: 19-Jul-2025].

[4] GeeksforGeeks, "Shortest Remaining Time First (Preemptive SJF) Scheduling Algorithm," GeeksforGeeks. [Online].
Available: https://www.geeksforgeeks.org/dsa/shortest-remaining-time-first-preemptive-sjf-scheduling-algorithm/.
[Accessed: 19-Jul-2025].