-------------------------------------------------------------------------------

1. Instructions

- replace any [...] with free text,

and

- replace the [?] with an X if you have completed that stage,

- replace the [?] with an \* if you have attempted that stage, but you know

it doesn't work completely; document why you think it doesn't work, plus

what you would do to fix the problem, at the end of the marksheet.

-------------------------------------------------------------------------------

2. Information

So that we can calibrate and improve the assignment in the future, give us

a rough idea how long (in hours) you spent on it \*in total\*:

effort : [140] hours

-------------------------------------------------------------------------------

3. Citation

Clearly it might have an influence on your mark, but the use of third-party

resources is allowed \*iff.\* it

- hasn't been explicitly prohibited by the assignment description, \*and\*

- is correctly cited.

Let us know any third-party source code or resources you used (if any) so

it's clear what's your work and what isn't:

I haven’t copied any code; however, I have researched into different types of buffers and came up with a circular buffer which is very similar to the one that I have linked below. It is a circular buffer; however, I didn’t see how I could change it to make it too different from the one presented on this webpage.

[https://www.embedded.com/ring-buffer-basics/]

-------------------------------------------------------------------------------

4. Marking

The following gives a stage-by-stage description of the assignment marking

scheme. Note this acts as an indicative guideline only, including weights

for each more obvious aspect (e.g., functional correctness); other aspects

outside this list can warrant an increase/decrease in marks, with examples

including hard to quantify features such as style, efficiency, robustness,

generality, or realism of a solution. Put another way, identifying then

reasoning about these latter aspects forms part of the assessment, so they

are not (necessarily) detailed explicitly.

Stage 1 : a baseline kernel

[X] - pre-emptive multi-tasking ( 30%)

[X] - priority-based scheduler ( 10%)

Stage 2 : closed generalisations and enhancements

[X] - fork, exec, and exit system calls ( 15%)

[X] - Inter-Process Communication (IPC) ( 15%)

Stage 3 : open generalisations and enhancements ( 30%)

[?] - MMU-based protection and virtualisation

\*OR\*

[?] - LCD screen and PS/2 device drivers and GUI

\*OR\*

[?] - file system based on simplified, emulated disk

\*OR\*

[?] - kernel port to real, physical hardware

------

(100%)

-------------------------------------------------------------------------------

5. Documentation

Any other documentation, notes or comments that you think are important or

might be easy to overlook (e.g., a subtle issue or technique in associated

source code) should go here:

[At the beginning of my philosophers you will notice a lot of comments. These comments have been done to explain how my program will work, and there I have explained how mutual exclusion was ensured and how starvation has been prevented. If there is something I missed out or you would like me to explain, please contact me at my university email.

Also, the waiting time of a round-robin scheduler is larger than the priority scheduler that I have created, which I believe makes my scheduler more effective. That means that if the processes ran using a round-robin scheduler, they would take a longer time to finish execution compared to my priority scheduler. I would choose a priority scheduler over a round-robin when there are many processes, as it might not make a big difference with a few processes running, however if there were a lot of processes, then having a long waiting time for each would be inefficient.

Also, in the “philosophers” function in philosophers.c, in the while(1) loop, after think(ID), I have included an EXIT call, which is so that after a philosopher has finished eating and thinking, it exits, so you can clearly see that all philosophers get to eat. However, this prevents the program to run forever, so in order to achieve that, just comment out the EXIT call after think(ID). You will realise that even though the EXIT call is called for all philosophers, there will still be a process running- that process is the waiter. ]

-------------------------------------------------------------------------------