CS3104 – Operating Systems

Assignment: P2 – Scheduling

Deadline: 16 Nov 2023 *Credits*: 50% of coursework mark

MMS is the definitive source for deadline and credit details

You are expected to have read and understood all the information in this specification and any accompanying documents at least a week before the deadline. You must contact the lecturer regarding any queries well in advance of the deadline.

Aim / Learning objectives

The purpose of this assignment is three-fold:

- to gain experience with kernel programming in C;
- to increase your understanding of scheduling algorithms;
- to give you hands-on experience of scheduling analysis.

Requirement One: Scheduling Algorithm Implementation

You are required to research and implement a scheduling algorithm for the provided toy operating system. The stub functions in `src/scheduler.c` must be implemented according to their documentation. The `kernel` directory contains pre-written code for the other components of the operating system and does not need to be touched. The code must build and run using the provided run script. You may not use any libraries, including the C standard library, however functions defined in the `kernel` directory can be used freely.

You can add new global variables and new members to the `sched_task` struct to track any required data. You will have to implement any datastructures that your chosen algorithm requires. A basic solution might add a `next` pointer to `sched_task` and a global `next_task` pointer to create a linked-list of tasks that are waiting to run.

The run script takes a single argument: a C program that will become the init program for the system. The operating system will be run in a virtual machine. Two example programs are provided in the `scenarios` directory that spawn child processes which test the scheduler. When all processes end the virtual machine will shut down.

Basic versions of `malloc` and `free` are available though you are advised that dynamic memory allocation is often avoided in kernel programming. A `printf` implementation is available, messages will appear on the virtual screen, on the simulator terminal and in the log file.

Your report should include a brief description of the theory behind your algorithm and evidence of testing.

Requirement Two: Analysis

The operating system emits log messages as it switches between processes. For example:

```
$$ timeslice summary for pid 5 (init 5) : queued at 7558872, ran at 7450837, ended at 7572180
```

Describes that pid 5, which is the init program, ran with argument 5, entered the scheduling queue at time 7558872, began running at 7450837 and ended at 7572180. All time values are in microseconds and are measured from some arbitrary point in time.

Use these lines to analyse the behaviour of your scheduling algorithm. The analysis should include computing some metrics of your choice based on wait and run times. A starter python script `analyse.py` is provided to help you get started, it shows how to extract the log data, compute a metric and plot a diagram. You don't have to start from this script if you don't want to. Strong submissions will write their own scenario files to evidence their claims.

The analysis should always be in service of answering relevant questions about scheduler behaviour, for example: are there cases where your algorithm starves some tasks of CPU time? How does it balance interactive and batch tasks? What are the pathological scenarios? Marks for this part are awarded for non-trivial, well-evidenced analysis in service of relevant and informative conclusions. The code quality of analysis scripts is not being assessed. **Dumps of data or figures are not, on their own, analysis.**

Getting started

Starter code can be found on studres.

Copy it to your practical folder and run it with "`sh run.sh scenarios/basic.c`". The operating system will start but immediately become idle. This is because the stub implementation of `dequeue_next_task` always returns `NULL`, meaning no processes are scheduled.

Keep in mind that C programs are hard to debug. Write small bits of code and check that they are correct before continuing. Some suggestions:

1) If an exception occurs, read the value of `%rip` (the instruction pointer) from the message and look it up in the file `build/kernel_disassembly.txt` to find the instruction and function where the error happened.
2) Ensure that you are using the correct format specifiers (in particular 64bit values require "`%ld`" or "`%lx`") and that the number of arguments to `printf` matches the number of template parameters.
3) Write a function `print_state()` that outputs a nicely formatted representation of the complete state of the scheduler. Make calls to `print_state` at appropriate points so you can see how the state is changing up to the point a problem occurs.
4) Include sanity checks in the `print_state` function and if something is inconsistent (for example the next and previous pointers on a doubly linked list don't agree) use the provided `crash()` function to stop the system at that point.

Submission

You are required to submit a zip file to the P2 slot on MMS. Your source code should all be contained in C files in the "src" directory. Your report should be in the ***PDF*** format. It must give an overview of your algorithm and explain its implementation and testing. The analysis and evaluation section should include any diagrams or tables you produced.

Assessment

Marking will follow the guidelines given in the school student handbook (see link at the end of this document). Some specific descriptors for this assignment are given below:

Mark range	Descriptor
1 - 6	A submission that does not compile or run with no analysis or a very limited report.
7 - 10	A partially working solution that compiles but which is too unstable and buggy

	for real use. Analysis that does not go far beyond the provided example or is highly unstructured or in poor style.
11 - 13	A mostly working implementation possibly containing minor bugs or limited evidence of testing. Analysis that shows some insight into the topic but may be unstructured or consist mainly of data dumps. Poorly written reports may also cause a submission to fall within this band.
14 - 16	A robust implementation of a basic algorithm, well documented and with good style alongside evidence of testing. Analysis that answers relevant questions with convincing evidence and in good style. All accompanied by a good report.
17 - 18	An implementation of a sophisticated algorithm with at most minor issues. Insightful analysis that synthesises theory and original experiment with excellent style. All accompanied by an excellent report. **OR** An implementation of a basic scheduling algorithm showing unusual clarity of design, robustly tested, and well described. Insightful analysis that synthesises theory and original experiment with excellent style. All accompanied by an excellent report
19 – 20	An implementation of a sophisticated scheduling algorithm showing unusual clarity of design, robustly tested, and well described. Complete and convincing analysis that shows a full grasp of the topic and extensive background reading. Original experiments are used to highlight the important properties of the chosen algorithm and the prose shows complete understanding of the relevant trade-offs. Accompanied by an exceptional, insightful report

Policies and Guidelines

Marking

See the standard mark descriptors in the School Student Handbook:

http://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_Descriptors

Lateness penalty

The standard penalty for late submission applies (Scheme A: 1 mark per 24-hour period, or part thereof):

http://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#lateness-penalties

Good academic practice

The University policy on Good Academic Practice applies:

https://www.st-andrews.ac.uk/students/rules/academicpractice/