<name>
<College>
<CRSID>

Diploma in Computer Science Project Proposal

A Testbed for Evaluating Scheduling Algorithms

<date>

Project Originator: This is a Model Project Proposal

Resources Required: See attached Project Resource Form

Project Supervisor: *<name>*

Signature:

Director of Studies: *<name>*

Signature:

Overseers: *<name>* and *<name>*

Signatures: *<no need to obtain Overseers' signatures yourself>*

<This Model Proposal describes a project which involves an investigation of scheduling algorithms. At various points in this document alternatives *are suggested or implied, and the submitted proposal is to be selected from the options given.>*

<For an example of a successful implementation of this project, see the 1996 Diploma Dissertation, A Test-Bed for Evaluating Scheduling Algorithms with Real-Time Systems*, by David M. Ingram.>*

Introduction and Description of the Work

Scheduling algorithms are widely used in communications networks and in operating systems to allocate resources to competing tasks. In operating systems, the scheduler must order the set of running programs for access to the CPU and other system resources. This project investigates several well known CPU scheduling algorithms by means of simulation, and compares their performance under different workloads.

In operating systems such as Unix, the scheduler maintains a list of the currently running tasks, which include user applications, such as xterm, emacs and the like, and "system tasks", such as the automount daemon, the X server etc. Each task is given the CPU for a short period of time, and the scheduler is responsible for making the decision as to which task to schedule next, given the constraints on the system, and the requirements of each task. I/O bound tasks typically execute relatively few instructions before performing I/O, but they must have frequent access to the CPU to provide a good response time to interactive users. CPU-bound tasks typically compute for a long period between bursts of I/O.

The aim of process scheduling is to assign the processor or processors to the set of processes in a way that meets system and user objectives such as response time, throughput or processor efficiency. In many systems the scheduling activity is broken down into three separate functions: long, medium and short-term scheduling which refer to the frequency with which scheduling decisions are taken. Long-term scheduling is used to decide what programs to admit to the system for execution, and is typically done by the user or batch system scheduler. Medium-term scheduling involves the swapping function: owing to the limited amount of physical memory on most systems, a program which has been suspended temporarily, for example awaiting I/O, may be written to disk; the medium-term scheduler keeps a list of tasks which are eligible to be swapped in to resume execution. Short-term scheduling may occur in response to system events, such as clock or device interrupts, system calls by tasks, signals and I/O interrupts. It is responsible for deciding which of the runnable tasks should receive the CPU next. This project will focus on a comparison of several short-term scheduling algorithms.

Various scheduling algorithms have been proposed; many are well understood. This project focusses on a comparison of schedulers, to enable a quantitative comparison of the performance of the different algorithms under a range of workloads. Schedulers can be either

work conserving or non-work conserving. Work conserving schedulers always schedule a task if one is runnable, whereas non-work conserving schedulers may idle the CPU even if there is a runnable task, typically to meet some performance constraint for the application, such as ensuring that it runs with a strict periodicity.

Examples of scheduling algorithms include First In First Out (FIFO), Last In First Out (LIFO), Processor Sharing (PS), Round Robin (RR), Earliest Deadline First (EDF), static Priority, Shortest Process First (SPF), and Shortest Remaining Time First (SRTF). The Unix scheduler computes dynamic priorities based on the history of a task and the system workload. Each algorithm has advantages and disadvantages, depending on the characteristics of the computational load on the machine. This project will investigate several of these algorithms by simulating their execution with a number of pre-defined workloads, contrasting their performance in each case.

Several performance criteria used to assess scheduling algorithms include:

Response time: typically the time from when the task is submitted until the first response is received. The system should aim to minimise this for all tasks, but individual tasks may have different response time requirements.

Timeliness: for algorithms which permit the user to attach deadlines to a task, this measures the proportion of time (or proportion of total scheduling epochs) when a task fails to meet its deadline.

Overhead: The proportion of time wasted due to computation of the schedule, and the system overhead due to context switching the tasks.

Predictability: A measure of the degree to which a task runs in a predictable manner (that is it takes approximately the same time or with the same cost), irrespective of the load on the system.

Efficiency: The proportion of time the system spends busy, given a specified workload.

Throughput: The number of tasks per second which the scheduler manages to complete.

Fairness: In the absence of user- or system-supplied criteria for selection, the scheduler should allocate a fair amount of the resource to each task.

Load balancing: The scheduler should balance the load across other system resources, such as memory, buffer usage, and I/O usage.

Simulation

Simulation is a technique used to capture the dynamics of complex systems by imitation. A simulation consists of a series of state space changes in a computer program that closely

follow the chronological order of events in the system being modelled. A simulation program is one which behaves like the system under study, evolving over time under the influence of external events and internal state modifications. In this project, the simulation program will emulate the behaviour of each of the scheduling algorithms chosen for the study.

Discrete event simulation concerns the modelling of a system as it evolves over time by a representation in which state variables change only at a countable number of points in time. These points indicate event occurrences. An example set of events for this project would be the occurrence of a timer interrupt, a request to do I/O by a task, completion of I/O and so on. During the execution of the simulation, the simulation program samples the relevant performance metrics of interest, such as the throughput, or response time of the scheduler under study. The output of the simulator is a set of observations of the performance of the system, perhaps as a distribution or simply a mean value. Of course, the simulation should be designed to be stochastic (that is to say, driven by random processes) to ensure that the results are representative of system performance over a range of parameters. The stochastic variability can be used to govern the arrival processes of events and tasks, and the sampling of results.

Resources Required

<The most suitable language will probably be an object-oriented language. Java or C++ would be ideal.>

Starting Point

<This is the place to declare any prior knowledge relevant to the project. For example any relevant courses taken prior to the start of the Diploma year.>

Substance and Structure of the Project

The aim of the project is to investigate the performance of several scheduling algorithms, producing quantitative comparisons of their behaviour under different loads. At least three algorithms must be compared: these may be selected from the list of schedulers below. At least one should be an implementation of a scheduler used in current operating systems. The simulation package should be extensible, to permit the implementation of additional scheduling algorithms should time permit. The simulator must include facilities for collection of performance measurements, and generation of random variates.

Possible schedulers which can be compared include:

1. The Unix scheduler: see the book by Leffler *et al.* on the 4.3BSD Unix Operating System.

2. The Windows NT scheduler: see Solomon's book on the Windows NT operating system.

3. A static priority scheduler: described in Stallings' introductory text on Operating Systems.

4. The Nemesis Scheduler: Nemesis is a multimedia OS developed at the Computer Laboratory; see for example Technical Report 376 by Timothy Roscoe (1995).

5. Hierarchical Round Robin: a rate-based server which ensures that each process gets a guaranteed rate of service in terms of CPU seconds per second of real time. It can also accommodate best effort tasks.

6. Static Priority Round Robin: divide processes into levels, and service each level using Round Robin; service levels in order of priority.

7. Preemptive Round Robin: rather than allowing a task its full quantum of time, pre-empt it from the CPU if it takes longer than a certain period of time.

8. Shortest Job First: always schedule the task with the shortest (anticipated) run time until its next I/O or completion.

9. Variants of the Feedback scheduler, which uses observations of application behaviour to estimate the time for which an application will run to order tasks for execution. The Unix scheduler fits into this category.

The project has the following main sections:

1. Familiarisation *<if need be>* with the programming language to be used and the surrounding software tools. Detailed design of data structures to used to represent the simulation state. Design of the interface between the simulator and the simulated scheduling algorithms. Choice of language may be important here: An object-oriented language will greatly assist the design of the system.

2. Development of the simulation toolkit which can be used to assess the performance of the various algorithms chosen. The toolkit will allow for discrete event simulation and will incorporate support for simulated events, simulated time, and a mechanism for recording performance measures for a simulated scheduling algorithm. At least two methods for generation of random variates from different distributions, for example the Poisson and Uniform distributions, must be incorporated.

3. Developing and testing the code for each of the three simulated scheduling algorithms. At least two of the algorithms must be used in current operating systems. The algorithms should be investigated by means of simulation. Code which schedules simulated applications according to the behaviour of each of the scheduling algorithms will be required. The scheduler should have a generic interface which would permit easy addition of further scheduling algorithms to the evaluation suite.

4. Evaluation of the scheduling algorithms: This will involve the development of a set of simulated applications whose activities should be broadly representative of typical applications on a multi-user operating system. They could, for example, include a (simulated) MPEG viewer, terminal session, editor session, ray-tracing application, system daemon, web browser, or compiler. At least three different simulated application types must be implemented. In any given run of the simulator, a number of applications of each type will be combined to form the system workload. A simple mechanism for specifying the workload should be incorporated. The workloads should combine different combinations of applications, and should represent high, medium and low total loads on the simulated scheduler.

5. Evaluation of the different scheduling algorithms using the simulator, and evaluation of the simulation toolkit which you have developed. Evaluation of the algorithms will involve the use of the simulator to measure the performance of each scheduling algorithm under different workloads, and tabulation or graphical presentation of the results. Evaluation of the simulation toolkit could consist of measurements of the performance of the simulator itself, with comments on the potential optimisation of the code.

6. Writing the Dissertation.

<You will have gathered that there is plenty of scope for the extension of this project. This includes the investigation of further scheduling algorithms, the improvement of the simulation environment, and evaluation of the selected algorithms with more widely varying workloads.>

<Additional scheduling algorithms can be implemented: consult your supervisor for pointers to the literature or to books which describe the algorithms in detail.>

<The simulator itself may be extended to facilitate the use of random variates from a wide range of distributions, or to permit the use of traces of events (for example from a measured system) as input to the system for event generation.>

<The workload used for the different scheduling algorithms should be carefully devised, in consultation with your supervisor. A typical characterisation of an application would include a measure of how long it typically holds the CPU, and whether or not it is I/O or CPU-bound. When a process blocks, depending on its activity, it can either become runnable immediately, wait for I/O to complete for a specified time, or wait for some other system event.>

Success Criteria

The following should be achieved:

- Implement a simulator
- Devise various workloads
- Compare at least three algorithms (one from a current operating system) for a number of runs of the simulator

<To do this, show results from running your simulator for at least three scheduling algorithms under various workloads.
For example, a table for each of high, medium and low load showing values of: response time, timeliness (when appropriate), overhead, efficiency, throughput, fairness for your algorithms.
Indicate from how many runs of the simulator the values in each table are obtained, and give mean values in the table with a deviation measure. Using all the results give the predictability of each algorithm.>

Timetable and Milestones

<In the following scheme, weeks are numbered so that the week starting on the day on which Project Proposals are handed in is Week 1. The year's timetable means that the deadline for submitting dissertations is in Week 34.>

<In the Project Proposal that you hand in, actual dates *should be used instead of week numbers and you should show how these dates relate to the periods in which lectures take place. Week 1 starts immediately after submission of the Project Proposal.>*

<The timetable and milestones given below refer to just one particular interpretation of this document. Even if you select exactly this interpretation you will need to review the suggested timetable and adjust the dates to allow as precisely as you can for the amount of programming and other related experience that you have at the start of the year. Take account of the dates you and your Supervisor will be working in Cambridge outside Lecture Term. Note that some candidates write the Introduction and Preparation chapters of their dissertations quite early in the year, while others will do all their writing in one burst near the end.>

Before Proposal submission

<This section will not appear in your Project Proposal.>

Submission of Phase 1 Report Form. Discussion with Overseers and Director of Studies. Allocation of and discussion with Project Supervisor, preliminary reading, writing Project Proposal. Discussion with Supervisor to arrange a schedule of regular meetings for obtaining support during the course of the year.

Milestones: Phase 1 Report Form (on the Monday immediately following the main Briefing Lecture), then a Project Proposal complete with as realistic a timetable as possible, approval from Overseers and confirmed availability of any special resources needed. Signatures from Supervisor and Director of Studies.

Weeks 1 to 5

<Real work on the project starts here (as distinct from just work on the proposal). A significant problem for Diploma candidates is that this critical period largely coincides with the Christmas vacation. There is no guarantee that supervisors will be available outside Lecture Term, but Diploma students take much less of a Christmas break than undergraduates do, and so have some opportunity for uninterrupted reading and initial practical work at this stage. It is important to have completed some serious work on the project before the pressures of the Lent Term become all too apparent.>

Study the ideas behind discrete event simulation, random variate generation and the collection of performance measures. Read up on scheduling algorithms and make sure you understand the ones you will simulate.

Milestones: Implement small (and they are small) bits of code which generate random variates of the selected distributions. Design structure of simulator.

Weeks 6 and 7

Further literature study and discussion with Supervisor to ensure that the chosen design of simulator is sound. Implementation of event management and clock in the simulator.

Milestones: Ability to deal with events in the basic simulation environment.

Weeks 8 to 10

Complete simulator by adding in code to measure the performance of a scheduling algorithm. Test simulator by implementing a trivial scheduling algorithm which acts first deterministically and then stochastically.

Weeks 11 and 12

Implement the three selected scheduling algorithms, and test their operation in the simulator using a trivial workload. Accumulate some statistics which show how the scheduling algorithm operates, and perhaps to trace the operation of the algorithm to facilitate debugging.

Weeks 13 to 19 (including Easter vacation)

Design and implement workload elements (simulated applications). Test simulated applications in the simulator. Develop a mechanism for specifying the set of simulated applications (the workload) to use in a particular run of the simulator. Write initial chapters of the Dissertation.

<The Easter break from lectures can provide a time to work on a substantial challenge such as the computation of logarithms, where an uninterrupted week can allow you to get to grips with a fairly complicated algorithm. This is a good time to put in some quiet work (while your Supervisor is busy on other things) writing the Preparation and Implementation chapters of the Dissertation. By this stage the form of the final implementation should be sufficiently clear that most of that chapter can be written, even if the code is incomplete. Describing clearly what the code will do can often be a way of sharpening your own understanding of how to implement it.>

Milestones: Preparation chapter of Dissertation complete, Implementation chapter at least half complete, code can perform a variety of interesting tasks and should be in a state that in the worst case it would satisfy the examiners with at most cosmetic adjustment.

Weeks 20 to 26

<Since your project is, by now, in fairly good shape there is a chance to use the immediate run-up to examinations to attend to small rationalisations and to implement things that are useful but fairly straightforward. It is generally not a good idea to drop all project work over the revision season; if you do, the code will feel amazingly unfamiliar when you return to it. Equally, first priority has to go to the examinations, so do not schedule anything too demanding on the project front here. The fact that the Implementation chapter of the Dissertation is in draft will mean that you should have a very clear view of the work that remains, and so can schedule it rationally.>

Work on the project will be kept ticking over during this period but undoubtedly the Easter Term lectures and examination revision will take priority.

Weeks 27 to 31

<Getting back to work after the examinations and May Week calls for discipline. Setting a timetable can help stiffen your resolve!>

Evaluation and testing. Finish off otherwise ragged parts of the code. Write the Introduction chapter and draft the Evaluation and Conclusions chapters of the Dissertation, complete the Implementation chapter. Evaluate the different scheduling algorithms by running the simulator for an extended period and collecting the results. Use a range of workloads and attempt to explain the behaviour of the different algorithms and the difference in their performance.

Milestones: Add the experimental data to the Evaluation Chapter of the dissertation, commenting on the performance of the different algorithms and the simulator itself. Dissertation essentially complete, with large sections of it proof-read by Supervisor and possibly friends and/or Director of Studies.

Weeks 32 and 33

Finish Dissertation, preparing diagrams for insertion. Review whole project, check the Dissertation, and spend a final few days on whatever is in greatest need of attention.

<In many cases, once a Dissertation is complete (but not before) it will become clear where the biggest weakness in the entire work is. In some cases this will be that some feature of the code has not been completed or debugged, in other cases it will be that more sample output is needed to show the project's capabilities on larger test cases. In yet other cases it will be that the Dissertation is not as neatly laid out or well written as would be ideal. There is much to be said for reserving a small amount of time right at the end of the project (when your skills are most developed) to put in a short but intense burst of work to try to improve matters. Doing this when the Dissertation is already complete is good: you have a clearly limited amount of time to work, and if your efforts fail you still have something to hand in! If you succeed you may be able to replace that paragraph where you apologise for not getting feature X working into a brief note observing that you can indeed do X as well as all the other things you have talked about.>

Week 34

<Aim to submit the dissertation at least a week before the deadline. Be ready to check whether you will be needed for a viva voce *examination.>*

Milestone: Submission of Dissertation.