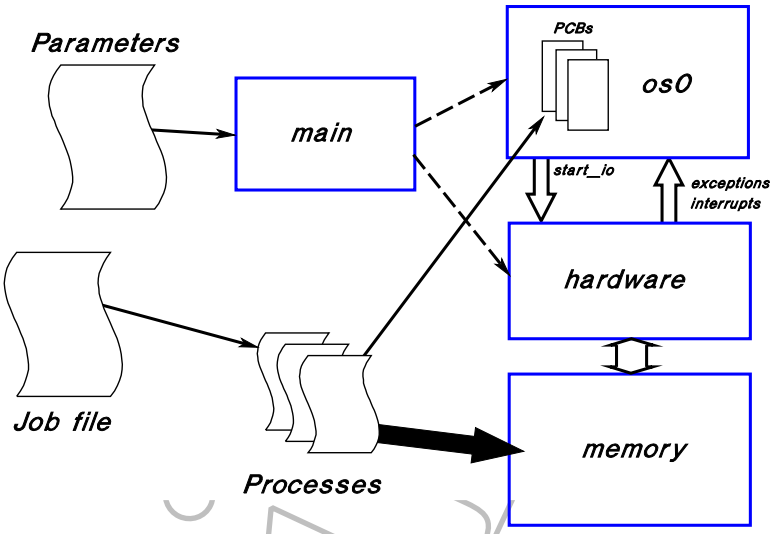
CS341 - Operating Systems
Operating System Simulator

Simulator Overview

This handout describes an operating system simulator that we will use for some process scheduling experiments.

The following diagram is an overview of the simulator. It consists of three main parts - the main program, the hardware (CPU) simulator, and the operating system simulator. Each of these parts is described in detail below:



The Main Program

The main program does most of the setup for the system, then starts the operating system and hardware simulators. After the simulation is over, it prints statistics about the behavior of the simulation. It consists of these function calls:

```
read_parameter_file(argc, argv);
Print_Welcome();
ipl_operating_system(argc, argv);
hardware();
Print_process_summary();
```

Here is a brief description of each of these functions.

- `read_parameter_file` - several parameters can be modified with appropriate entries in a parameter file. This routine reads the parameter file, parses the entries, and sets the appropriate parameter values. The possible parameters are:
  - `SCHEDULER` (default value is 3) Use `SCHEDULER` to set the process scheduler algorithm:
    - 1 to select First-come First-serve (FCFS)
    - 2 to select Shortest Job First (SJF)
    - 3 to select Round-Robin (RR)
  - `TIMESLICE` (default value is 5) If round robin scheduling is used, use `TIMESLICE` to set the time slice that will be used.

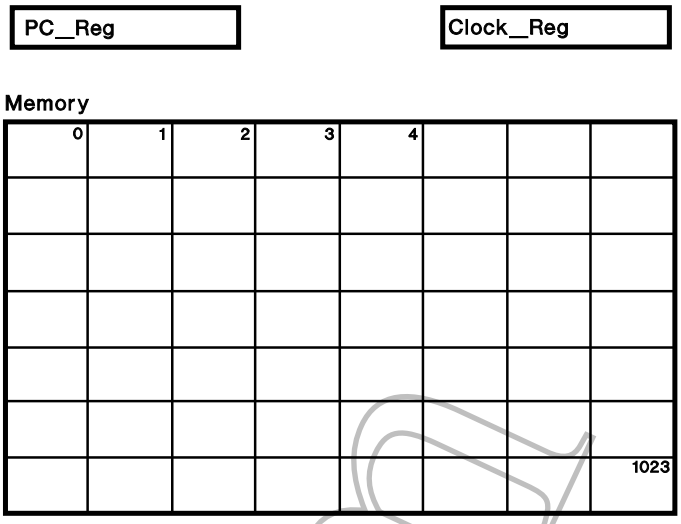
- INSTRUCTIONS (default value is 1) Setting this parameter to 1 causes the instructions to be printed as they are executed by the hardware. Tested and used inside the function Print_instruction (see hardware0.c).
- TRANSITIONS (default value is 1) Setting this parameter to 1 causes a message about the transitions between states to be printed. It is called within all the transition functions of the operating system. Tested and used inside the function Print_transition (see os0main.c).
- PCB (default value is 0) Setting this parameter to 1 causes the entire PCB to be printed at the start of every Dispatch transition. Tested and used inside the function Print_PCB_information (see os0main.c).
- MEMORY (default value is 0) Setting this parameter to 1 causes the contents of memory to be printed after all the processes have been created. Tested and used inside the function Print_memory (see os0main.c).
- READYQ (default value is 0) Setting this parameter to 1 causes the ready queue to be printed at the start of every Dispatch transition. Tested and used inside the function Print_ReadyQ (see os0main.c).
- INTERRUPTS (default value is 0) Setting this parameter to 1 causes the list of pending future interrupts to be displayed every time a new interrupt is added. Tested and used inside the function Print_interrupt_list (see hardware0.c).
- WELCOME (default value is 1) Setting this parameter to 1 causes a welcome message and a list of parameters to be listed at the start of the simulation. Tested and used inside the function Print_Welcome (see os0main.c).
- SUMMARY (default value is 1) Setting this parameter to 1 causes an accounting summary to be listed at the end of the simulation. Tested and used inside the function Print_summary_process (see os0main.c).

If the name of the parameter file is DEBUG, then no actual file is read but instead all of the print debug flags are turned on. If the name of the parameter file is DEBUGOFF, then no file is read but instead all of the print debug flags are turned off, except for WELCOME and SUMMARY. These options are available because sometimes during code debugging it is helpful to see everything that is happening within the system or to see only the summary report at the very end.

- Print_Welcome - displays a description of the current operating system and some of the important parameters being used.
- ipl_operating_system - reads the job file (a list of the processes that will be run). Each of the processes referenced in the job file are created. Finally, one of the processes is Dispatched.
- hardware - This is the heart of the simulation. It increments the Clock_Reg, executes an instruction, and checks to see if any interrupts will occur at this time. The loop is repeated until all processes complete, or until it completes 100,000 iterations.
- Print_process_summary - This routine prints the statistics involved with the simulation: how long each process was in various states, average ready times, etc. The summary is printed only if the SUMMARY parameter is set to 1.

Hardware Simulator

The hardware simulator models a very simple CPU that has memory (where the programs will reside), a program counter (called `PC_Reg`) to keep track of the next instruction to execute, and a clock (called `Clock_Reg`) to keep track of time and to generate interrupts at intervals so that processes can share the CPU. Here is a small diagram that shows these parts of the hardware:



The instructions that this CPU can execute are very simple - there are only three:

- **COMPUTE** (opcode = 1). This simulates some general computation.
- **HALT** (opcode = 0). This halts the process, moving it to the Terminated state.
- **NOP** (opcode = 2). This does nothing. It is here in case a distinction must be made between doing "real" work (using the **COMPUTE** instruction) and doing nothing.

A "program" for this computer is a file with numbers representing these opcodes. You can think of it as a compiled executable module. For example, a file might contain the numbers 1 1 1 1 0. When this process is executed it would do four **COMPUTE** instructions and then the **HALT** instruction - at which time it would be done.

The basic operation of the hardware simulator is to execute an instruction in the memory location pointed at by `PC_Reg`, then check whether there are any interrupts or exceptions that might require handling. This loop is repeated until all the processes have made their way to the `Terminated` state. When all processes are done, the simulation stops.

In this version there is one hardware function called by the operating system:

```
start_io(int type, int process);
```

Whenever the TIMER must be set so that an interrupt can be scheduled for some future time, the hardware makes a call to `start_io`. This function is also used to signal the end of the simulation by calling `start_io(OFF,OFF)`.

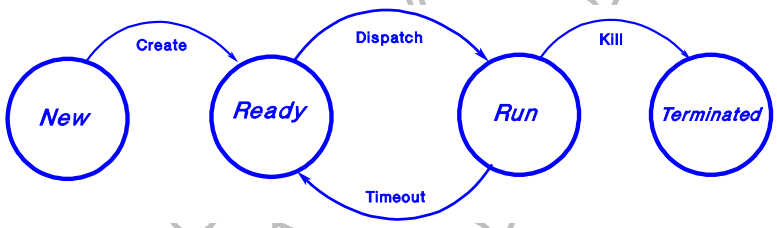
The hardware simulator is expecting that the following functions exist within the Operating System simulator:

- `exceptions(int type);` - deals with exceptional conditions such as illegal instructions or a halt instruction. Exceptions generally cause the current process to be replaced by another process, sometimes with an error message.
- `interrupts(int process);` - currently, the only kind of interrupt that can occur is one generated by the TIMER. The TIMER interrupt occurs when a running process has run out of its time slice (time slices are used only when we have a round robin process scheduling algorithm). The current process is sent to the Ready state (via the Timeout transition) and a new process needs to be Dispatched.

The Operating System Simulator

The operating system simulator is responsible for handling interrupts and exceptions, and for doing process scheduling. The scheduling algorithm that is used is determined from the `parameters` file read by the main program.

Like a "real" operating system, the simulator maintains a process control block for each process in the system; it implements the process state diagram shown below. It differs from a real OS in that it doesn't actually run on the hardware simulator, but only manages the hardware by manipulating processes and responding to interrupts.



Process Control Block

All the information about a process is stored in the process control block (PCB):

`PC`	-	local program counter for this processi
`state`	-	current state of the process
`st_time`	-	time the process started in current state
`run`	-	total time the process has been in the RUN state
`ready`	-	total time the process has been in the READY state
`loc`	-	the memory location of the start of the process
`size`	-	the size of the process in memory
`name`	-	name of the process (up to 30 characters)
`next`	-	pointer to the next PCB

Interrupts

There is one type of interrupt: TIMER. The hardware simulates interrupts with a linked list of future interrupt events. Currently this list contains a single TIMER event, but a linked list is used so future interrupts can be added. Each element in this list consists of the time the event will occur, the type of event, and the process associated with the event.

TIMER events are scheduled by the `Dispatch_process` routine by a call to `start_io(TIMER, current_pid)`. The interrupt for this event will occur after a time specified by `TIMESLICE`. These interrupts are only used if round robin scheduling is being used.

Operating System Functions

The following functions make up the operating system itself:

- `exceptions` (also described above) - deals with exceptional conditions such as illegal instructions or a halt instruction. Exceptions generally cause the current process to be replaced by another process, sometimes with an error message.
- `interrupts` (also described above) - In the current operating system, the only kind of interrupt that can occur is one generated by the TIMER. The TIMER interrupt occurs when a running process has run out of its time slice (time slices are used only when using round robin). The current process is sent to the Ready state (via the Timeout transition) and a new process needs to be Dispatched.
- `Dispatch_process` - determines which process is to be scheduled next, and updates the PCB of this process. It also calls `start_io(TIMER, pid)` to create a future TIMER interrupt if round robin is being used. `Print_transition` is called to display a message about the transition.

  `Dispatch_process` handles one special case: when all processes are in the Terminated state, it stops the simulation by calling `start_io(OFF,OFF)`.
- `Timeout_process` - changes appropriate PCB info, adds the current running process to the ready queue, and calls `Print_transition` to announce the transition.
- `Kill_process` - changes appropriate PCB infoi, updates the number of processes still left in the system, and calls `Print_transition` to display a message about the transition.

Simulator code and sample job files

There are three primary pieces of code that make up the system:

- The hardware simulator (`hardware0.c`) - this is the code for all aspects of the hardware simulation. A header file contains the constants and structure definitions used in the hardware simulator.
- The "operating system" routines (`os0.c`) - this file contains the scheduler and other supporting functions which perform the main functions of the operating system.
- Main Program code (`os0main.c`) - this is the main program for the simulator, along with a few supporting functions that do file input and parsing, and initialization of the system. A header file contains the constants and structure definitions used in `os0main.c`.

A tar file (`os0.tar`) contains all the `.c` and `.h` files necessary to build os0. A `makefile` is included that will build the os0 system. A few sample data files are included that provide a fairly complete set of test examples. A `README` file explains each test. To install the package:

1. Download the tar file to the directory where you will do your work.
2. Untar the code files with the command `tar -xvf os0.tar` . This will create a subdirectory called `os0` containing all the files.
3. To create the executable, type `make`. This will compile the files in the proper sequence and create the simulator in a file called `os0run`.

Example run

Below is a small example of a complete set of input files. The job file contains the name of only one process. The total memory required by all the processes cannot exceed 1024.

The file `job0b1` contains:

```
pro0b
```

The file `pro0b` contains:

```
1 1 1 0
```

The data in the job and process files is free format (each entry is separated by white space).

Sample Output

```
> os0run job0b1 parameters
Welcome to the CS Computer System model 341 (version xxxx.xx)
Computer configuration:
  MEMORY_SIZE: 1024
  SCHEDULER (1=FCFS,2=SJF,3=RR): 3
  TIMESLICE: 5
Time=0 Pid=0 transition=Create.
Time=0 Pid=1 transition=Create.
Time=0 Pid=1 transition=Dispatch.
Time=1 Pid=1 instruction=INST PC=1
Time=2 Pid=1 instruction=INST PC=2
Time=3 Pid=1 instruction=INST PC=3
Time=4 Pid=1 instruction=HALT PC=4
Time=4 Pid=1 transition=Kill.

Final Summary of Process Statistics (time=4)
PID State Run Ready Total Name
 1    T     4    0     4  pro0b2

Average Run     Ready    Total times
       4.00     0.00     4.00
```