OS ASSIGNEMENT

1. Implementation

1.1 TIMEINT Module

TIMEINT is the module which provides the interrupt handler of the interval timer interrupt, named timeint_handler(). It's responsibility is to reset the timer to the interval value, forcing it to start counting again. To set the timer value, we use the set_timer() function provided. The argument of this function is the Quantum variable, which is the interval of the timer.

1.2 INT Module

The INT module must provide the general interrupt handler, gen_int_handler(). When an interrupt occurs, control is passed to the general interrupt handler, which must call the appropriate specific interrupt handler, according to the event that must be handled (e.g.: timeint_handler is invoked when the interval timer interrupts). This functionality is provided with a switch-case statement, that selects the appropriate handler based on the value of the cause field of the interrupt vector.

Also, the general interrupt controller triggers the CPU scheduling algorithm, calling the dispatch() function. Before the dispatch, the last_cpuburst and accumulated_cpu fields of the currently running process are updated.

Finally, nested interrupts should be handled. A dispatch within a nested interrupt doesn't make sense, because the outermost dispatch will be taken into consideration, cancelling the nested ones. To avoid rescheduling in nested interrupts, a static counter is used, that counts the nest level. This is done by incrementing the counter when an invocation of the general interrupt handler occurs, and decrementing when exiting the handler. Update of the current process and rescheduling are performed when we are at the outermost interrupt handling.

1.3 CPU Scheduling Module

The CPU module must provide three processes, insert_ready, which accepts a PCB as an argument and adds it to the ready queue, dispatch which reschedules and switched the running process and cpu_init which is called when the simulator "boots" and executes initialization code for the CPU module.

First of all, an implementation of the ready queue and of the selected scheduling algorithm must be created. The ready queue was implemented as a single linked list, of nodes which have a pointer to a PCB. Also, the following queue procedures where implemented:

- ready_queue_is_empty – checks if the ready queue has any nodes. Returns BOOL
- insert_pcb – creates a new node with the provided PCB, and inserts it in the ready queue. The insertion is done in the tail of the queue.

- pcb_exists – searches all the ready queue for a match with the given PCB. The equality comparison is preformed over the PCB ID. Returns BOOL

The scheduling algorithm that was selected is a priority based algorithm. Each process starts on creation with a base priority. When a dispatch happens, the priority of the running process is updated. If the dispatch was because the quantum of the process was expired, the priority is lowered. If it was because of an I/O blocking, the priority is incremented. The process that will use CPU next, is the one with the highest priority. If there are many processes with the highest priority, a FCFS (First Come, First Served) policy is used to select between them.

For the task of finding the node in the ready queue with the maximum priority, the get_max_priority_pcb function was written. This function traverses the linked list starting from the head and holding the node with the maximum priority found until the current node. When it has traversed the queue, returns the PCB with the maximum priority found, de-queues the node and frees the memory hold by the node. The fact that new nodes are inserted in the end of the queue, implements the FCFS policy when equality occurs, because the first maximum that is found by get_max_priority_pcb, will be kept, until only a highest priority PCB is found.

The cpu_init function, first resets the interval timer, and then initializes the ready queue to empty queue. The insert_ready function must insert the given PCB argument in the ready queue. First a check is done to ensure that the PCB is not already in the queue, using pcb_exists. Next, the processes priority is updated, with the rules that are mentioned before. Finally the PCB status is set to 'ready' and the PCB is inserted using insert_pcb.

The dispatch function is responsible for the rescheduling and task switching of the CPU. It is invoked when an interrupt occurs. First, it has to remove the running process and to insert it in the ready queue to avoid CPU starvation of this process. This is done using insert_ready. Next, rescheduling takes place to find which process will be using the CPU next. In this case, get_max_priority_pcb is used. Finally the process must be allocated the CPU. For that purpose the PTBR variable is redirected to point to the processes page table. Also the status of the process must change to 'running'. Additionally the last_dispatch filed must be updated to the current clock using get_clock. Also prepaging is used to avoid page faults. Finally the interval timer is reseted.

2. Simulation Results

All the simulations where executed with no errors or warnings. In xariseach simulation, snapshots of the system are created and a table with a summary of statistics is presented for each snapshot. The statistics informs us about CPU and memory utilization, throughout, average CPU time per process, average waiting time per process and average turnaround time per process.

After simulating the modules, the simulation results were compared with the results of '*OSP.demo*'. For the CPU module the comparison is shown in the next table:

	This implementation	**Demo**
CPU Utilization	37,78%	36,77%
Memory Utilization	70,89%	74,56%
System Throughput	6,899	5,4978
Average CPU time/process	62,9214	63,4924
Average Waiting time/process	931,0929	915,1439
Average Turnaround time/process	988,0143	974,3561

From the above table we can see that CPU, memory utilization and system throughput is slightly better in our implementation. But the average CPU, waiting and turnaround time per process are slightly worse. The differences at the statistics are because of the different CPU scheduling algorithms used. For the other two modules, this statistics are the same for this implementation and the demo.