Report

Answer 1. Ready list is data structure which holds the processes which are in ready state for execution and is sorted according to descending priority. System calls that operate on it directly are the ones that call the ready() method. Ready() is called by

1. Send
2. Signal
3. Signaln
4. Resume

Answer 2. By default, Xinu uses preemptive priority-based scheduling. This policy could lead to starvation as a low priority process may not be scheduled at all as long as we have a long running high priority process, or if there is a continuous stream of high priority processes that may prevent the low priority process from being scheduled at all.

Answer 3. Resched decides whether to put the process in the readylist based on following decisions. If the process is in state current and has priority less than the first process in the readylist then the process is added to the readylist i.e if a higher priority process is available in the readylist only then the current process is put back to the readylist.

Answer 4. Resched is triggered from following events.

1. Ready – when a function is put in ready state by the ready() function.
2. Suspend – A process is put in suspended state and resched() is called if the process being suspended is the current process.
3. Yield – Whenever syscall yield is called resched is triggered.
4. Recvtime – Resched is triggered when process is put in sleepqueue and state is set to RECTIM.
5. Sleep - Sleep triggers reched after it puts current process to sleep.
6. Receive – The current process is put in receive state and then resched is triggered.
7. Kill – When kill is called resched is triggered.
8. Clkhandler- resched is triggered when the preempt value goes to zero.
9. Wait – resched is called after the process is put on semaphore queue.

CT1 Code is stored in file print_ready_list.c.

CT2 runtime is updated in clkhandler.c, turnaroundtime is updated in kill.c and num_ctxsw in resched.c

CT3 and **CT5** are in create.c User processes are assigned priority 1.

CT4 is done in resched.c

Lottery Schedule

Code description

Files modified

include/ clock.h, kernel.h, process.h, prototypes.h, queue.h

system/ clkhandler.c, clkinit.c, create.c, initialize.c, kill.c, main.c, lottery_sched.c, print_ready_list.c, ready.c, resched.c

Flow.

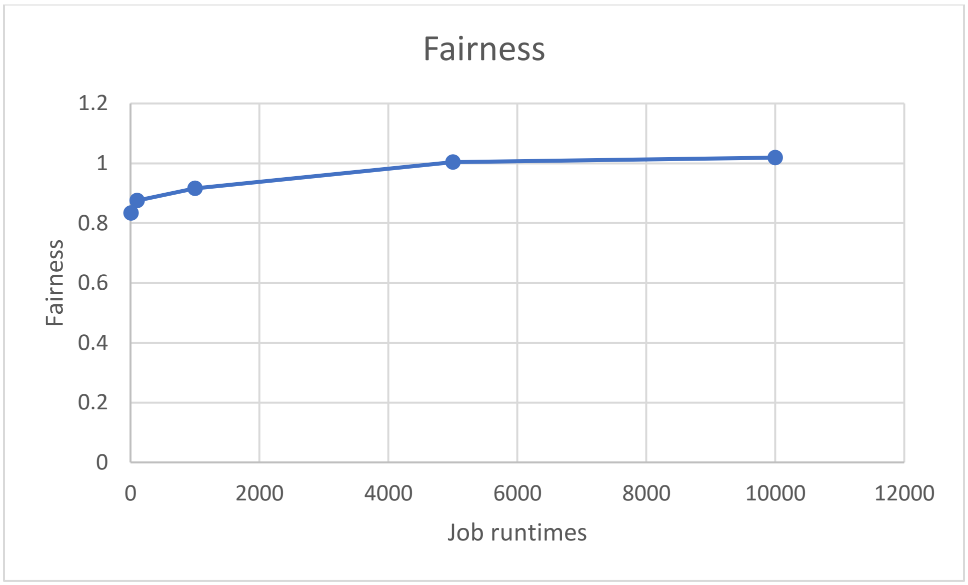
1. When resched is called I first check whether the current process is system process and has a higher priority than the first process in the readylist and the current process is not nullprocess. If above conditions are met context switch does not happen and the current process remains the current.
2. If above condition is not met and current process is system process put it in readylist
3. Else put it in userprocesslist. The insert_to_user method inserts the the userprocess according to the number of tickets in descending order and if tickets are the same then according to ascending order of pid. Within the method the total_tickets global variable is updated.
4. Now we check if firstid of readylist is not the nullprocess and whether the userprocesslist is nonempty. If yes then the user_flag is set.
5. We call recompute_tickets method to update the total_tickets variable. This is done to ensure that if set_tickets is called after the process is already put in userprocesslist.
6. If user flag is set after step 4 then we check whether the firstid and lastid of the userprocesslist are same. If yes it indicates that only one process is present in the userprocesslist and no need to call lottery_scheduler. The process is dequeued from userprocesslist and context switch happens. Else if more than one process is present then findwinner method is called with a random number as the generated parameter. We traverse the userprocesslist process by process and updates a counter with the number of tickets. The process for which counter is greater than the random number, that process is dequeued and set as the current process.
7. If user flag is not set then first process in readylist (nullprocess) is scheduled.

Analysis of fairness

In main.c I set two process with runtimes (10, 100, 1000, 5000 and 10000) and calculated the fairness by dividing the turnaround time of first process by the second process. Values are found below in the table

Job runtime	Turnaround time for Job1	Turnaround time for Job2	Fairness
10	50	60	0.833333
100	210	240	0.875
1000	1870	2042	0.915769
5000	10044	10004	1.003998
10000	20040	19670	1.01881

Note: for last two processes first process finished after the second hence the fairness values are more than 1.



Analysis: The fairness approaches 1 as the runtimes increase. This is because one process will finish one quantum away from the other. The one quantum has a very high impact on fairness if job times are lesser and hence as job runtimes run higher the effect of one quantum is negligible and hence fairness approaches 1.

MLFQ Scheduling

Code Explanation:

Files modified

include/ clock.h, kernel.h, process.h, prototypes.h, queue.h, resched.h

system/ clkhandler.c, clkinit.c, create.c, initialize.c, kill.c, main.c, mlfq.c, print_ready_list.c, ready.c, resched.c

Flow

1. When resched is called I first check whether the current process is system process and has a higher priority than the first process in the readylist and the current process is not nullprocess. If above conditions are met context switch does not happen and the current process remains the current.
2. If process is a system process then simply set it too ready and put it to ready list. If it is a user process and if it has completed its allotted time then decrement its queue to med or low and reset its time allotment. Then insert it in appropriate queue by calling insert_to_user which enqueues processes based on its ptype value (1-> high, 2-> med, 3->low)
3. If priority boost timer has exceeded priority boost period then dequeue process from med and low (in that order) and put them to high priority queue by calling priority_boost queue.
4. Next we check if readylist is nonempty and firstid of readylist is not the nullprocess then set user_flag to False and dequeue the first process in readylist.
5. Else check if there is a process present in high, med and low queues (in that order), If a process is present then dequeue it and schedule it. If process is not present in any of these queues then schedule nullprocess.
6. Now set the correct time slice to make the correct scheduling decisions and call ctxsw.