CS 423 MP2

Group Members:
1) Furquan Shaikh(fmshaik2)
2) Gurmeet Singh(gurmeet2)
3) Puneet Chandra(pchandr2)

Implementation and Design Decisions:

(A) Initialization:
Following tasks are performed during mp2 kernel module initialization:

1) On init, mp2_kernel_mod creates a proc directory entry "mp2" and a proc entry under this directory "status". Thus, user processes can use the entry /proc/mp2/status to read from or write to.

2) Head for a link list which will hold a list of all the registered processes is initialized. Another head for the run queue is initialized.

3) A kernel thread "mp2_sched_thread" is created which acts as the dispatcher thread for selecting the next process that is to be scheduled. This is the heart of the scheduling process.

4) A semaphore mp2_sem is initialized to control access to the linked list. This is required since both kernel module and kernel thread can make updates to the linked list. Hence, synchronization is achieved between them using semaphore mp2_sem.

(B) Proc entry updates:

1) When a write is made onto the proc entry -
User data is copied into kernel buffer and depending upon the command passed by the user (R,Y,D), appropriate functions are called (Registration, Yield, Deregistration)

2) When a read is made on the proc entry -
Scan the linked list and send the details of each registered process present. Details include PID, P and C.

(C) Process control block:

1) Additions are made to the process control block to adapt it for the mp2 scheduler.

2) Every registered process is represented by mp2_task_struct. This structure contains the following fields:
a) PID
b) Pointer to task_struct for this process
c) Wakeup Timer to track release time

d) Computation Time for real-time loop
e) Period
f) List head member for list of registered processes
g) List head member for run queue
h) Next release time for this process
i) State of the process -- READY, RUNNING or SLEEPING

(D) Registration Process:

1) It first parses user sent data to obtain PID, P and C.

2) It calls admission control to check if this process can be registered.

3) Then, it allocates an mp2_task_struct for the process and does the initialization. Initial state of the process is marked as SLEEPING and next release time is set to jiffies + period.

4) The process is then added to the list of registered processes.

(E) Admission control:

1) Here, the total CPU utilization is calculated for all the registered processes and the new process which wishes to be registered.

2) Since kernel does not support floating point operations, the total utilization is calculated in multiple of 1000 and compared with 693.

3) If the total utilization of all registered process including the new process is less than 693, then the process is admitted, else rejected.

(F) De-registration:

1) Once the process is completed with the real time loop, it will send a deregister command.

2) The kernel module will find the mp2_task_struct for the calling process. It is removed from the runqueue and also deleted from the list of registered processes. Cleanup is performed for the process by freeing all memory allocated for it, stopping the timer and resetting the priority of process to normal.

(G) Wakeup Timer Handler:

1) It identifies the mp2_task_struct of the process for which the timer expired.

2) Marks the state of the process as READY and adds it to the run queue according to priority.

3) It then wakes up the dispatcher thread.

(H) Yield:

1) It first finds out the mp2_task_struct using pid, if the currentl is not the same as the one calling yield(can happen when the process calls yield for the first time).

2) It checks if the next_release_time has already passed.
a) If yes, check if the process is in sleeping state(can happen again when the process calls yield for first time). In this case add the process to runqueue. Wake up the dispatcher thread.

b) If no, set a timer for amount of release_time. Remove the process from runqueue. Move it to sleeping state. Wake up the dispatcher thread.

(I) Dispatcher Thread:

1) This thread sleeps until some event wakes it up. (eg: process calling yield, wakeup time expiry, process deregistration)

2) If run queue is not empty, it selects the first process since that is the process having highest priority. It checks if there is any currently running process and determines which has a higher priority. If currently running process has a higher priority, it passes control back to it. If the process in run queue has a higher priority, the currently running process is pre-empted and moved back to ready state.

3) Priority of new process is set to real time priority. mp2_current is updated to point to the currently running process. Next release time of current process is calculated and control is given to that process.

4) When the kernel module calls a kthread_stop, this thread exits from the while loop.

(J) Run Queue:

1) The run queue is used to keep track of all the processes that move to ready state.

2) Processes are added to this queue in priority order determined by their period.

3) This allows the dispatcher thread to select the next process to be scheduled in O(1) time complexity.

4) The processes are added to the run queue in O(n) time complexity. If n increases too much, the linked list can be replaced by a red-black tree which will give a time complexity of O(log n). Currently, the run queue is implemented using a linked list.

(K) Clean up:

On exit, entire cleanup is performed:

1) Free all allocated memory

2) Kill scheduler thread

3) Free all list heads

4) Delete proc entries

(L) User App:

It is implemented as per the specifications provided in the MP. There are two options available in user app. Command line arguments can be passed for setting a particular P,C and n value. If no arguments are passed it selects a random number and chooses P, C and n from a static array. (n is the factorial to be calculated in real time loop)

Testing:
The following tests have been performed successfully:
1) Execute user app with utilization much less than 0.693.
2) Execute user app with utilization very close to 0.693 (Some periods are missed)
3) Execute user app with utilization greater than 0.693 (Registration fails)
4) Execute 2 user apps in such a way that both move to ready state at same time (Pre-emption observed in kernel)
5) Run 3 or 4 processes with different P and C values at the same time.

Submission Files:
1) mp2_kernel_mod.c: Kernel module implementation of mp2
2) mp2_user_app.c: User app implementation of mp2
3) mp2_given.h: Header file containing given code
4) Makefile: Makefile to compile kernel module and user app

Compilation Steps:
make clean
make

How to use:
1) insmod mp2_kernel_mod.ko
2) ./mp2_user_app
3) Output messages can be seen on screen
4) ./mp2_user_app 1000 300 10
5) rmmod mp2_kernel_mod

Multiple instances of mp2_user_app can be executed to check for multiple processes