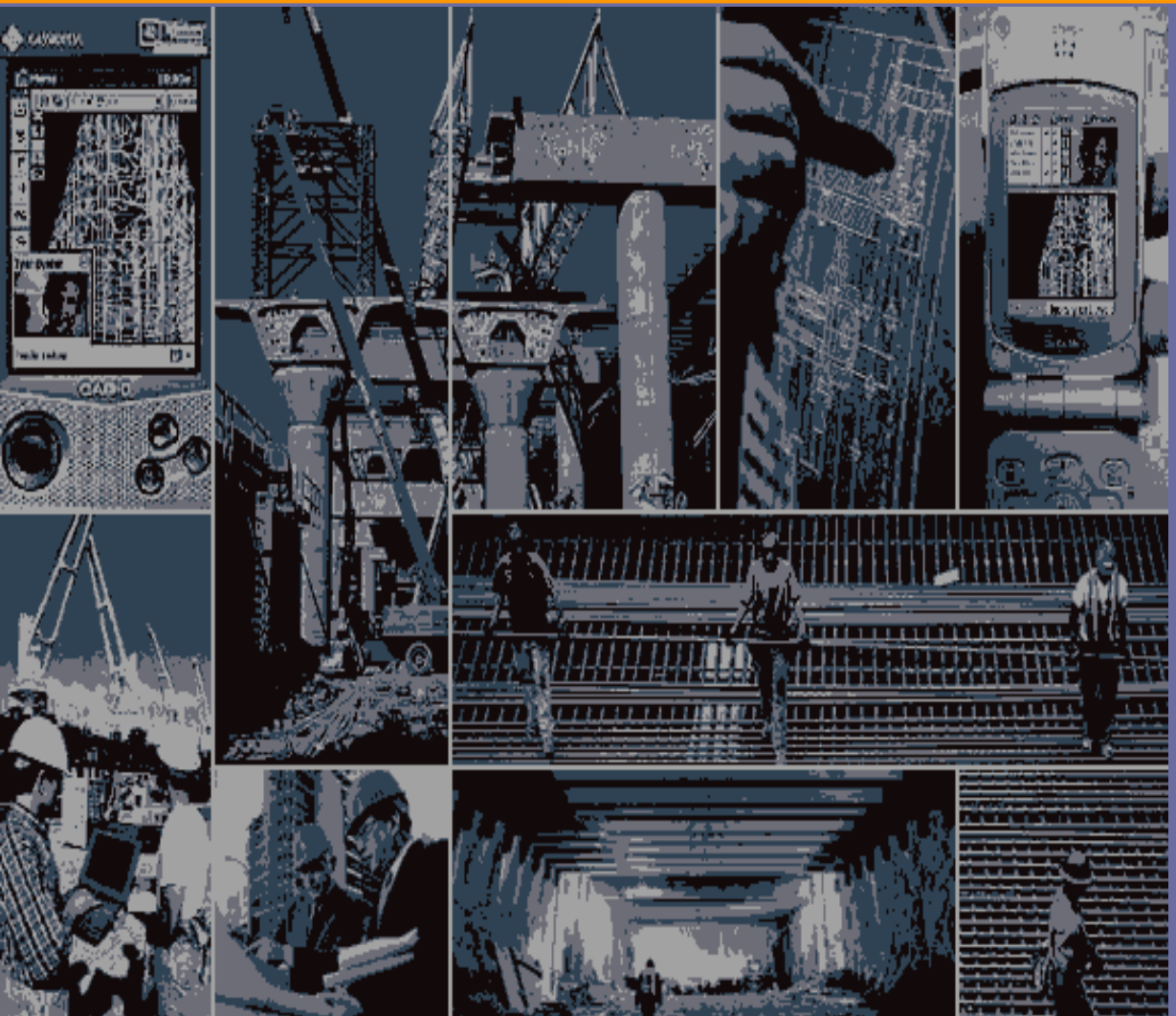
CS423: Operating System Design – Machine Problem 2 (MP2)

Rate-Monotonic CPU Scheduling

September 21, 2012



Reza Shiftehfar

PhD. Candidate and
Research Assistant in Computer Science
Dept. of Computer Science
University of Illinois at Urbana Champaign
E-mail: sshifte2@illinois.edu

MP2: Rate-Monotonic CPU Scheduling

Announcements:

- MP2 is out.
- Any group with missing person can email me TODAY.

Mp2 Submission :

- Due on 10/3 midnight
- Submit on Compass.
  - If that doesn't work, email me your files: sshifte2@illinois.edu
- Make Snapshot before deadline
- Have to demo your code and answer questions
- Signup sheet will be up next Monday (Sept. 24th)
  - First come, first serve for taking the available times

MP2: Rate-Monotonic CPU Scheduling

MP2 goals:

- MP1 was focused on getting to know available tools
- MP2 is focused on developing useful Kernel code
- Develop a Rate Monotonic Scheduler
- Develop a bound-based Admission Control for our scheduler

MP2 similarity to MP1:

- Develop scheduler as a Linux Kernel Module (LKM)
- Use Proc File System to communicate with user space application
  - Proc/mp2/status

MP2: Rate-Monotonic CPU Scheduling

Real-Time Scheduling/System:

- Scheduling/System that can
  - guarantee a response time
  - within the strict time constraint

Categories:

- Hard Real-time
  - Any deviation from deadline results in failure
  - E.g. Medical systems such as pace maker, airline navigation system
- Soft Real-time
  - Some misses of deadlines are tolerable
  - Degrade the system's Quality of Service
  - E.g. Software updating flight plans for commercial airliners

MP2: Rate-Monotonic CPU Scheduling

Other categories:

- Firm Real-time
  - Stricter than soft real-time
  - Only Infrequent deadline misses is tolerable
  - Live video-audio system
- Imprecise Real-time
  - Each task has 2 part:
    - mandatory part (hard real-time),
    - optional part (soft real-time)

MP2: Rate-Monotonic CPU Scheduling

Real-Time Scheduling Algorithm:

- Static Scheduling
  - Scheduler has complete knowledge of tasks:
    - Tasks coming
    - Deadlines
    - computation time
    - future releases
    - Precedence constraints
- Rate Monotonic (RM) scheduling is the best solution

MP2: Rate-Monotonic CPU Scheduling

Real-Time Scheduling Algorithm:

- Dynamic Scheduling
  - No complete knowledge of the tasks and their constraints
  - Arrival of new tasks can be:
    - with unknown duration
    - At unknown time
    - With unknown constraints
  - Scheduler can only schedules the current set of tasks
- There are 2 main types for Dynamic Scheduling:
  - Resource Sufficient Environment
  - Resource In-Sufficient Environment

MP2: Rate-Monotonic CPU Scheduling

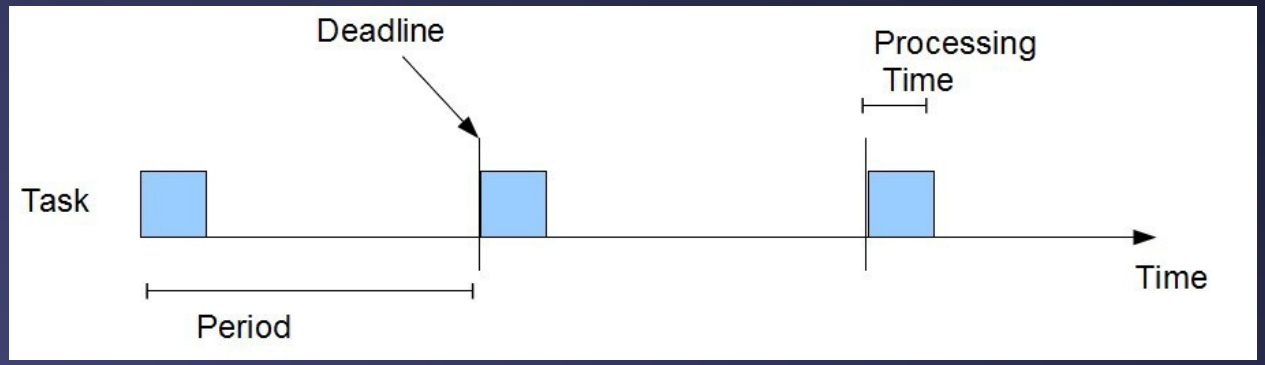
Real-Time Scheduling Algorithm:

- Dynamic Scheduling
  - Resource Sufficient Environment
    - System resources are sufficient to a priori guarantee
    - New unknown tasks might arrive at any time
      - But always enough resources exists to schedule them
    - Earliest Deadline First (EDF) algorithm is the optima solution
  - Resource In-Sufficient Environment
    - No guarantee on the sufficiency of the existing resources
    - We might face rejection/delaying of some of the tasks
    - Not suitable for Hard Real-Time environments

MP2: Rate-Monotonic CPU Scheduling

Real-Time Scheduling Assumptions:

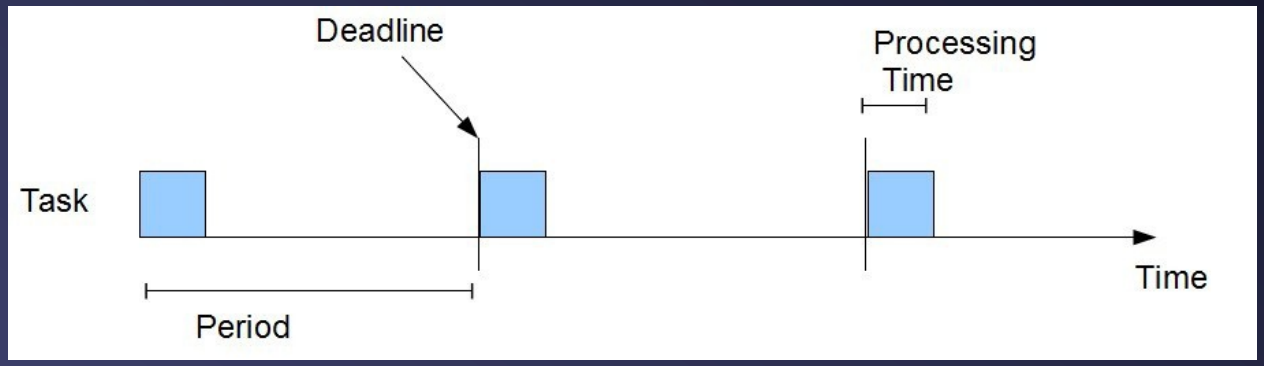
- Tasks are periodic => constant intervals between requests
- Each task must be completed before the next request for it occurs
- Tasks are independent
  - Request for a certain task does not depend on the initiation/completion of requests for other tasks
- Run-time of each task is constant



MP2: Rate-Monotonic CPU Scheduling

Real-Time Scheduling :: Naming Convention:

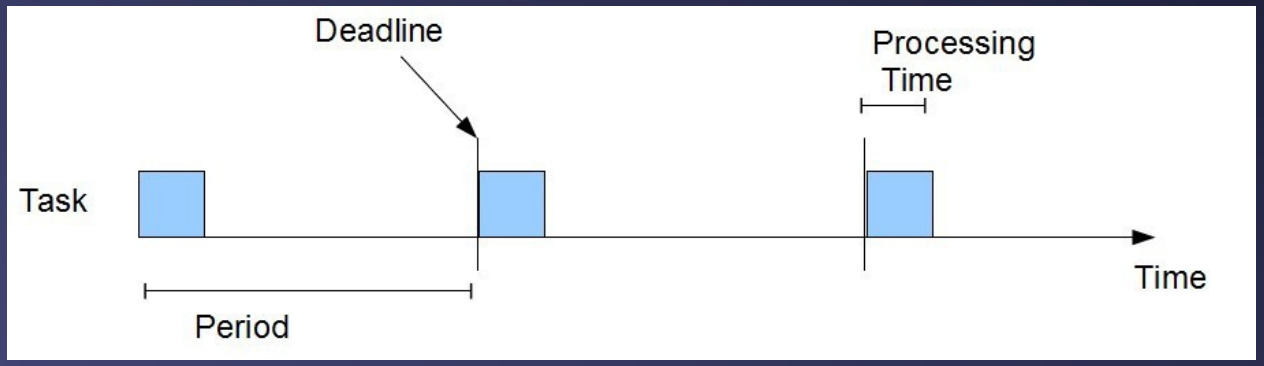
- Task i => τ_i
- Request period of T_i
- Execution time of C_i



MP2: Rate-Monotonic CPU Scheduling

Real-Time Scheduling :: Rate Monotonic Scheduling (Static Scheduling)

- Assign higher priority to tasks with higher request rate (smaller T_i)
- This is done regardless of their execution time (C_i)
- At any time, pick the task with highest priority and execute it
- Priority of tasks does not change over time
- Least upper bound to processor utilization is 70% for large task sets
  - Utilization of more than 70% can still be achieved with suitably selected tasks



MP2: Rate-Monotonic CPU Scheduling

Real-Time Scheduling :: Earliest Deadline First Scheduling (Dynamic Scheduling)

- Assign task priorities based on their deadline
  - Task with earliest deadline has highest priority
- Priority of tasks changes with time
- Capable of reaching full processor utilization
- It is feasible if and only if:

$$\sum_{i=1}^{m} \frac{C_i}{T_i} \leq 1$$

MP2: Rate-Monotonic CPU Scheduling

MP2 Overview:

- Goal: Design a Rate-Monotonic Scheduler with Admission Control module
- Scheduler should allow the following:
  - Registration
    - with desired parameters (PID, Period, Computation Time)
  - Yield
  - De-Registration
- Scheduler only register a process if it passes through Admission Control
  - Admission control checks if the parameters will lead to a feasible schedule
- Simple Test Application.

MP2: Rate-Monotonic CPU Scheduling

MP2 Description:

- Rely on Linux Scheduler to perform context switches
- Reading /proc/mp2/status should give :
  - List of registered applications
  - Scheduling parameters of each registered application

MP2: Rate-Monotonic CPU Scheduling

MP2 Description:

- Test Application:
  - Single threaded periodic application
  - Factorial application is a good choice
  - Register itself with the scheduler through Admission Control
    - Specifies its pid, period, processing time
  - After registration, read proc filesystem entry to ensure that it is accepted
  - Signal the scheduler that it is ready by sending a Yield message
  - Initiate the Real-Time loop
  - At the end, de-register itself

MP2: Rate-Monotonic CPU Scheduling

MP2 Description:

- Test Application:

```
void main (void)
{
        REGISTER(PID, Period, ProcessTime); //Proc filesystem
        list=READ STATUS(); //Proc filesystem: Verify the process was admitted
        if (!process in the list) exit 1;
        //setup everything needed for real-time loop: t0=gettimeofday() for test.c
        YIELD(PID); //Proc filesystem
        //this is the real-time loop
        while(exist jobs)
        {
                do_job(); //wakeup_time=t0-gettimeofday() and factorial computation
                YIELD(PID); //Proc filesystem
        }
        UNREGISTER(PID); //Proc filesystem
}
```

MP2: Rate-Monotonic CPU Scheduling

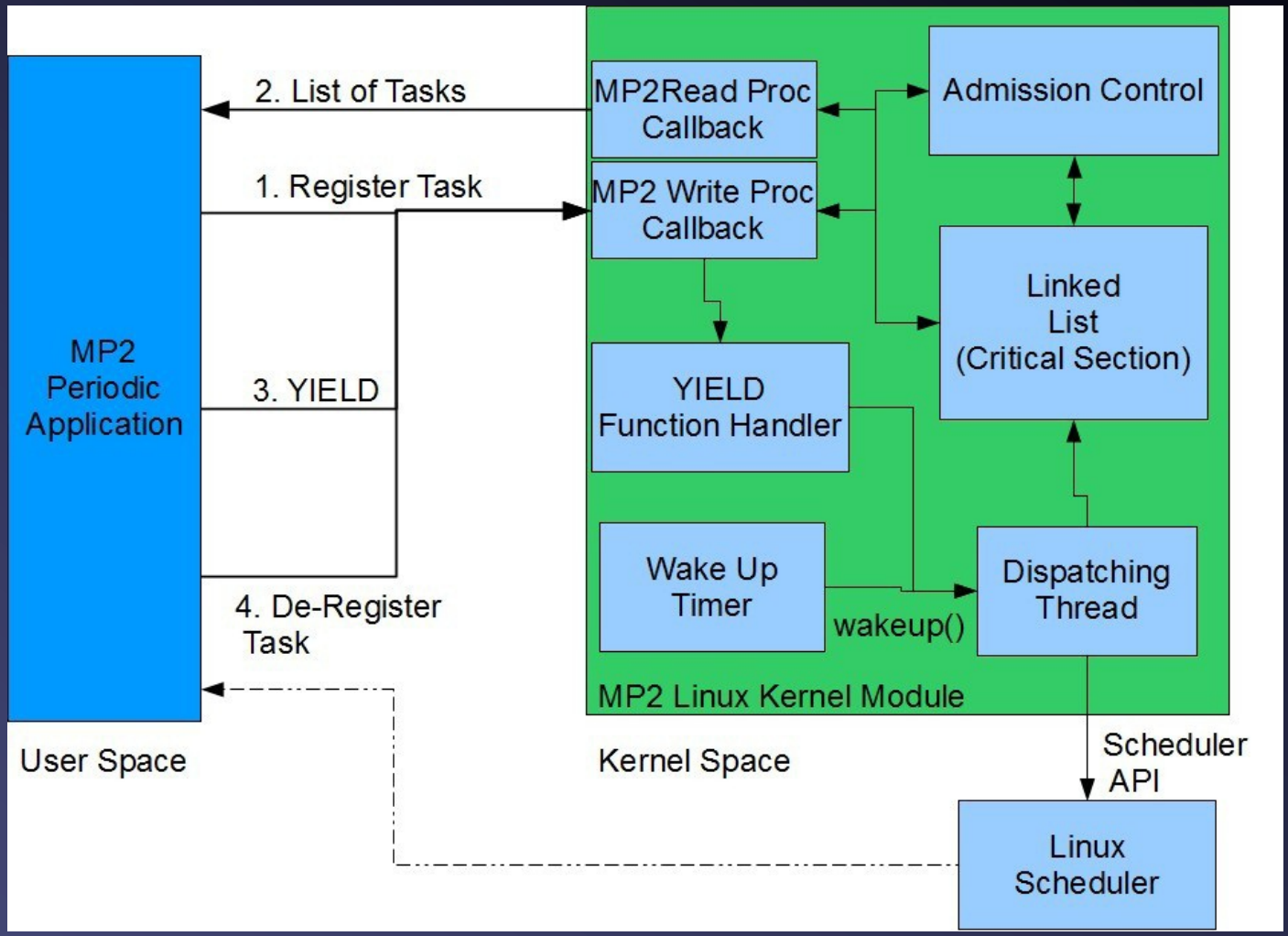
MP2 Description:

- Test Application:
  - An application with higher priority will preempt application with lower priority as soon as it becomes available to run
  - Application that has finished its current job will Yield the CPU
    - This is done through proc file system entry
    - At this time, CPU will schedule the next application with highest priority

MP2: Rate-Monotonic CPU Scheduling

MP2 Description:

- Overview of the process of MP2:



MP2: Rate-Monotonic CPU Scheduling

MP2 hints:

- Registration/Yield/De-Registration is through proc file system:
  - Use the first character to detect actions:
    - For registration: “R, PID, PERIOD, COMPUTATION”
    - For YIELD: “Y, PID”
    - For DE-Registration: “D, PID”
- You need a dispatching thread:
  - Goes through the list of registered processes
  - From the list of tasks with READY state, picks the one with highest priority
    - This is the task with the shortest period
  - Preempts the currently running task (if any)
  - Context switches to the chosen task

MP2: Rate-Monotonic CPU Scheduling

MP2 hints:

- For Context switching:
  - We use Linux Scheduler API for this
  - Notice:
    - Any task running on the SCHED_FIFO will hold CPU for as long as it needs
    - We can trigger a context switch by using SCHED_SETSECHEDULER()
  - For the new running task, dispatching thread does this:

    wake_up_process():

    Set it's sched_priority = MAX_USER_RT_PRIO-1;

    sched_setscheduler(task, SCHED_FIFO, &sparam);
  - For the old running task (preempted task), dispatching thread does this:

    Set it's sched_priority=0;

    sched_setscheduler(task, SCHED_NORMAL, &sparam);

MP2: Rate-Monotonic CPU Scheduling

MP2 hints:

- Admission control Module:
  - *Checks to see if adding the new task leads to a schedulable set of tasks*
  - It it is not schedulable, the new task is rejected
  - A task set is schedulable if:

$$\sum_{i \in T} \frac{C_i}{P_i} \leq 0.693$$

    - This must hold after adding the new task
    - T is the set of all tasks including the new task to be admitted
    - C is the processing time of each task
    - P is the period of each task

MP2: Rate-Monotonic CPU Scheduling

MP2 grading:

Criterion	Points
Read and Write Proc filesystem callbacks parsing the three scheduler messages (Registration, Yield, De-Registration) and printing the status*	10
Correct Implementation of the Process Control Block*	5
Correct Implementation of the Registration Function with proper Admission Control*	10
Correct Implementation of the De-Registration Function*	5
Correct Implementation of the Wake-Up Timer *	5
Correct Implementation of the YIELD Function*	15
Correct Implementation of the Scheduling Thread*	15
Correct Initialization and De-Allocation of data Structures*	5
Test application follows the Application Model (Registration, Initial Yield Message, Real-Time Loop, De-Registration)*	10
Document Describing the implementation details and design decisions	10
Your code compiles and run correctly	5
Your code is well commented, readable and follows the software engineering principles listed in this hand-out	5
TOTAL	**100**

Linux Kernel Programming

Conclusion:

- *For your demo:*
  - *Be there at least 10 minute ahead of time*
  - *Bring a laptop*
  - *Have everything prepared before your scheduled time*
- *In general:*
  - Allow 2 days to respond to your emails
  - Send follow up if you don't hear from us after 2 days
- *Seek help*
  - *Discuss with your group*
  - *Discuss on Piazza*
  - *Come to office hours*
- *Special thanks to* Raoul Rivas from Uni. Of Illinois