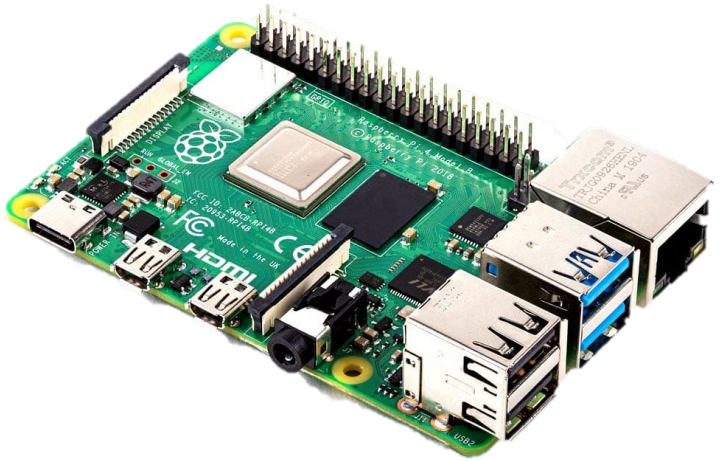
ECE 1175 Embedded Systems Design

Lab 4 – RMS, EDF and Priority Inheritance



ECE 1175 – Lab 4

- **Scheduling Algorithms**
  - Rate Monotonic Scheduling (RMS)
  - Earliest Deadline First Scheduling (EDF)
  - Lab task 1: scheduling simulation

- **Priority Inheritance**
  - Multithreading in C
  - Lab task 2: demonstration of priority inheritance

Recap RMS

- **Rate Monotonic Scheduling (RMS)**
  - Higher rate (1/period) => Higher priority
  - Preemptive
  - Schedulability check

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq U_b(n) = n\left(2^{1/n} - 1\right)$$

U: CPU utilization

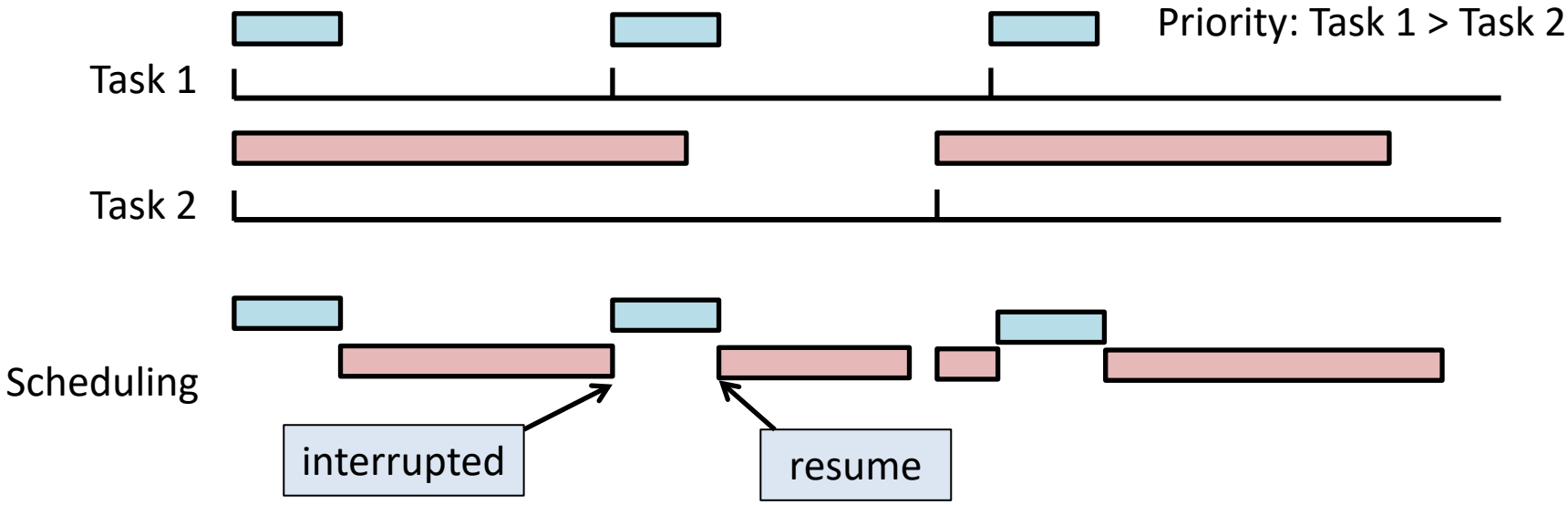
C_i: execution time of process i

P_i: period of process i

  - $U_b(2) = 0.8284$

Recap RMS

- **Preemption**
  - The ability of the operating system to **interrupt** and temporarily suspend the execution of a running task to give way to another task



Recap EDF

- **Earliest Deadline First Scheduling (EDF)**
  - Earlier absolute deadline => Higher priority
  - Preemptive
  - Schedulability check

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$$

Lab Task 1

- **Simulate RMS and EDF based on the code template *rms.c***
  - To schedule two tasks
  - Implement schedulability check and preemption
  - Implement EDF in a separate file edf.c

- **Check-off**
  - Run your code with different task settings
    - Schedulability check
    - Correctness with preemption
  - Submit your code to Canvas

Lab Task 1

- **Example1 – RMS, T1 = (10, 25), T2 = (15, 60)**

```
                    ---------------------------------------
                    Rate Monotonic Schedule (RMS) Algorithm
                    ---------------------------------------
please input period and execution for A process
default: 25, 10: 25 10
please input period and execution for B process
default: 60, 15: 60 15

simulation started
when T=0, process A0 and B0 are generated together
when T=0, program switched to run process A0!
when T=10, process A0 is done
when T=10, program switched to run process B0!
when T=25, process B0 is done
when T=25, process A1 is generated
when T=25, program switched to run process A1!
when T=35, process A1 is done
when T=50, process A2 is generated
when T=50, program switched to run process A2!
when T=60, process A2 is done
when T=60, process B1 is generated
when T=60, program switched to run process B1!
when T=75, process B1 is done
when T=75, process A3 is generated
when T=75, program switched to run process A3!
when T=85, process A3 is done
when T=100, process A4 is generated
when T=100, program switched to run process A4!
```

You can verify the correctness of the code by running these examples or by manually drawing the flow chart

Lab Task 1

- **Example2 – RMS, T1 = (10, 20), T2 = (15, 30)**

```
                        Rate Monotonic Schedule (RMS) Algorithm
                        ---------------------------------------
please input period and execution for A process
default: 25, 10: 20 10
please input period and execution for B process
default: 60, 15: 30 15
CPU Utilization : 1.00

simulation started
when T=0, process A0 and B0 are generated together
when T=0, program switched to run process A0!
when T=10, process A0 is done
when T=10, program switched to run process B0!
when T=20, process A1 is generated
when T=20, program switched to run process A1!
process B missed deadline! not schedulable
```

1. Your program should check if the task set is schedulable or not (schedulability check)

Lab Task 1

- **Example3 – RMS, T1 = (20, 50), T2 = (35, 100)**

```
                    Rate Monotonic Schedule (RMS) Algorithm
                    ---------------------------------------
please input period and execution for A process
default: 25, 10: 50 20
please input period and execution for B process
default: 60, 15: 100 35
CPU Utilization : 0.75

simulation started
when T=0, process A0 and B0 are generated together
when T=0, program switched to run process A0!
when T=20, process A0 is done
when T=20, program switched to run process B0!
when T=50, process A1 is generated
when T=50, program switched to run process A1!
when T=70, process A1 is done
when T=70, program switched to run process B0!
when T=75, process B0 is done
when T=100, process A2 and B1 are generated together
when T=100, program switched to run process A2!
when T=120, process A2 is done
when T=120, program switched to run process B1!
when T=150, process A3 is generated
```

2. Your program should be able to handle preemption

Multithreading in C

- Basis
  - Thread
  - Mutual Exclusion (mutex)
  - Scheduling (FIFO)
- Priority Inheritance

Multithreading in C - thread

- **Thread creation**

```
include <pthread.h>

void *thread1handler() {  // thread handler function
    // do some work
}

int main() {
    // declare a thread variable
    pthread_t thread1;
    // create a thread
    pthread_create(&thread1, NULL, thread1handler, NULL);
    // wait for thread to complete
    pthread_join(thread1, NULL);
}
```

Multithreading in C - mutex

- **Mutual Exclusion (mutex)**
  - A synchronization mechanism that ensures only one thread can access a critical section of code or a shared resource at a time

Multithreading in C - mutex

```
include <pthread.h>

pthread_mutex_t lock; // define mutex

void *thread1handler() {  // thread handler function
    pthread_mutex_lock(&lock);
    // critical section: do some work
    pthread_mutex_unlock(&lock);
}

int main() {
    // initialize mutex
    pthread_mutex_init(&lock, NULL);
    // create thread
    pthread_t thread1;
    pthread_create(&thread1, NULL, thread1handler, NULL);
    pthread_join(thread1, NULL);
}
```

Multithreading in C - mutex

- To enable priority inheritance

```
include <pthread.h>

pthread_mutex_t lock; // define mutex
pthread_mutexattr_t attributes; // define attributes

int main() {
    // initialize mutex
    pthread_mutexattr_setprotocol(
        &attributes, PTHREAD_PRIO_INHERIT
    );
    pthread_mutex_init(&lock, NULL);
    // create threads
    ...
}
```

Multithreading in C - scheduling

- **Thread scheduling**

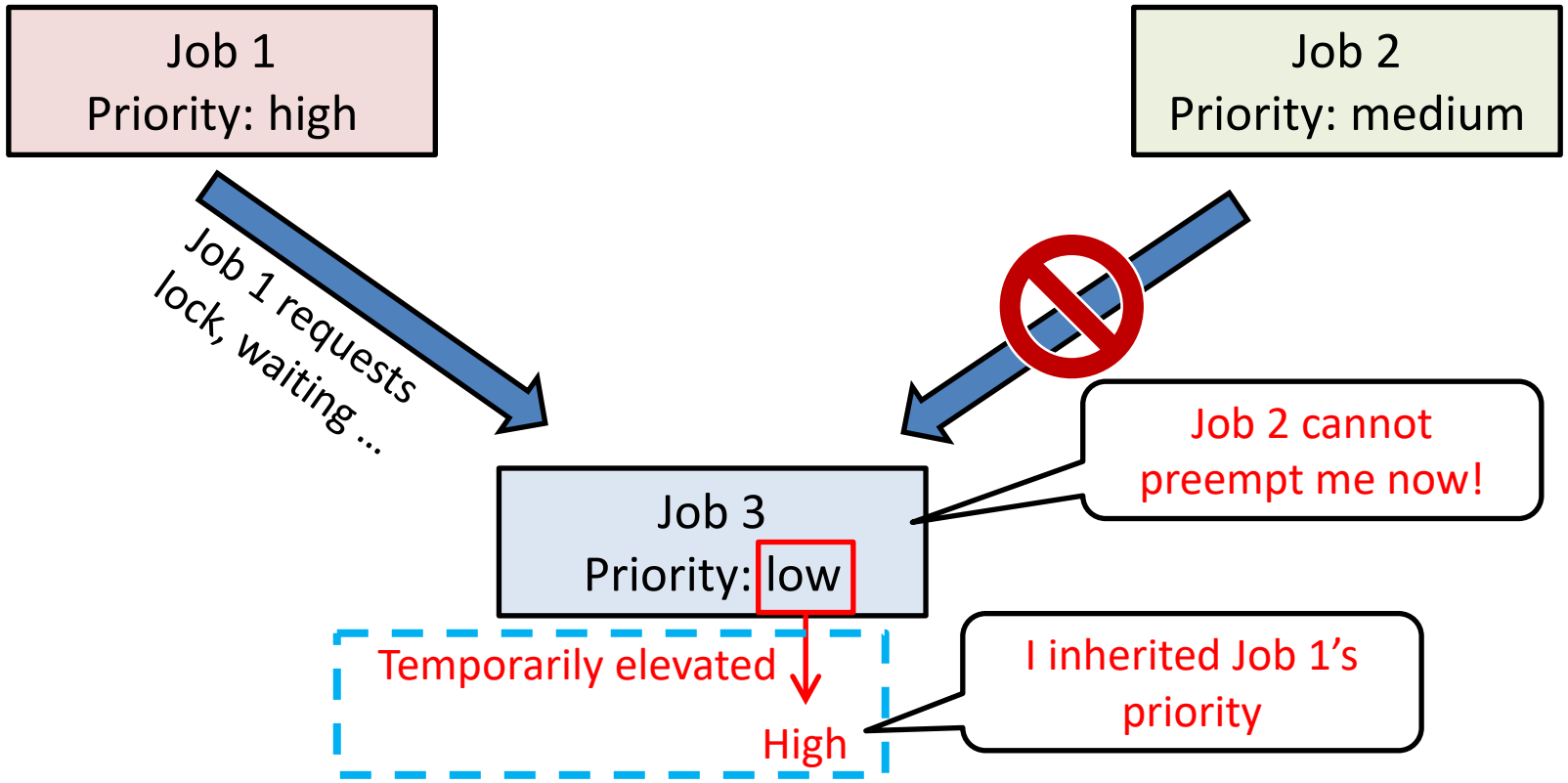
```
include <pthread.h>
include <sched.h>
int policy = SCHED_FIFO; // define scheduling policy

int main() {
    // define a schedule parameter
    struct sched_param param1;
    int priority1 = 10; // set an integer for priority
    param1.sched_priority = priority1;
    // bind the schedule parameter to a thread
    pthread_attr_t attr1;
    pthread_attr_init(&attr1);
    pthread_attr_setschedpolicy(&attr1, policy);
    pthread_attr_setschedparam(&attr1, &param1);
    pthread_create(&thread1, &attr1, function1, NULL);
}
```

Recap Priority Inheritance

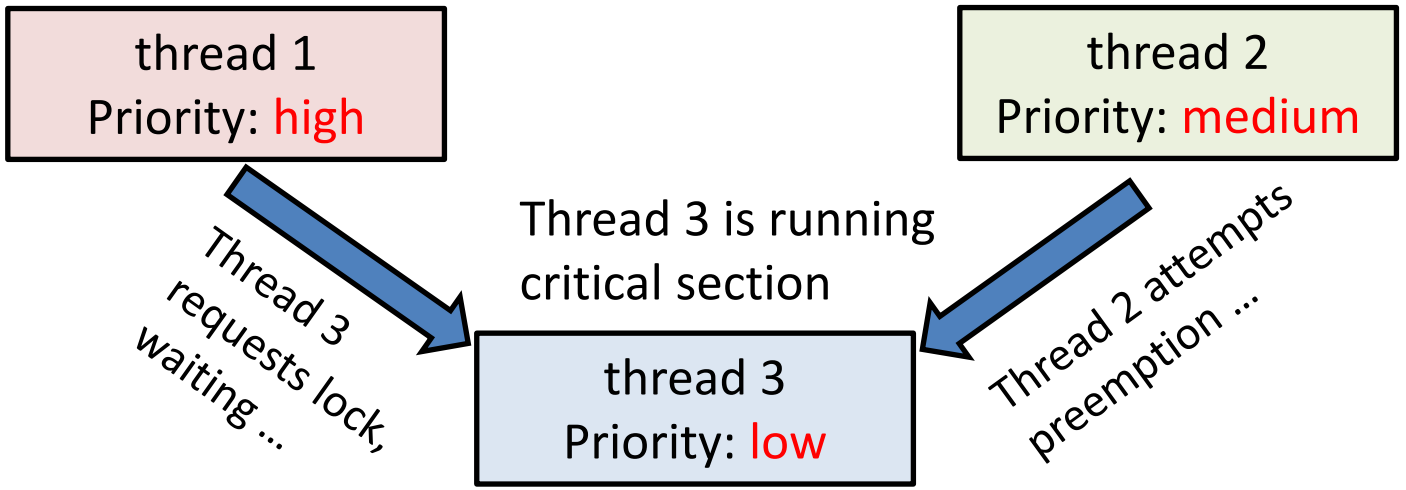
- **Priority Inheritance**

Job 3 is running critical section (hold lock)

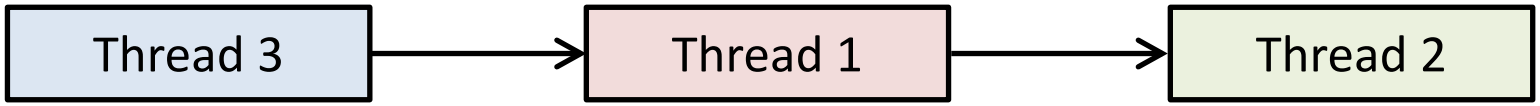


Lab Task 2

- **Write multithreading C code to demonstrate priority inheritance**



Threads should be created in this order



Lab Task 2

- **Example**

```
Creating thread3...
Thread 3 starts
Thread 3 requests the lock
Thread 3 had the lock
Thread 3 running, priority 1, process 10%...
Thread 3 running, priority 1, process 20%...
Creating thread1...
Thread 1 starts
Thread 1 requests the lock
Thread 3 running, priority 10, process 30%...
Thread 3 running, priority 10, process 40%...
Thread 3 running, priority 10, process 50%...
Creating thread2...
Thread 3 running, priority 10, process 60%...
Thread 3 running, priority 10, process 70%...
Thread 3 running, priority 10, process 80%...
Thread 3 running, priority 10, process 90%...
Thread 3 released the lock
Thread 1 had the lock
Thread 1 running, priority 10, process 10%...
Thread 1 running, priority 10, process 20%...
Thread 1 running, priority 10, process 30%...
Thread 1 running, priority 10, process 40%...
Thread 1 running, priority 10, process 50%...
Thread 1 running, priority 10, process 60%...
Thread 1 running, priority 10, process 70%...
Thread 1 running, priority 10, process 80%...
Thread 1 running, priority 10, process 90%...
Thread 1 released the lock
Thread 1 complete
Thread 2 starts
Thread 2 running, priority 5, process 10%...
Thread 2 running, priority 5, process 20%...
Thread 2 running, priority 5, process 30%...
Thread 2 running, priority 5, process 40%...
Thread 2 running, priority 5, process 50%...
Thread 2 running, priority 5, process 60%...
Thread 2 running, priority 5, process 70%...
Thread 2 running, priority 5, process 80%...
Thread 2 running, priority 5, process 90%...
Thread 2 complete
Thread 3 complete
```

Priority Set: Thread1=10, Thread2=5, Thread3=1

Thread execution order: 3 -> 1 -> 2

Thread 1 requested the lock but it was hold by thread 3. Thread 3 continued, but priority elevated (**priority inheritance**)

Thread 2 tried to preempt thread 3 but failed because of the priority inheritance

Thread 1 got the lock after thread 3 released it

Thread 2 preempted thread 3 after thread 1 completed

Thread 3 completed after thread 2 completed

Lab Task 2

- **Important Notes**
  - Link to pthread lib when compiling the code

    gcc -pthread program.c -o program

  - Make sure to run the program on a single CPU

    sudo taskset -c 0 ./program

  - Try *pthread_example.c* to create a thread and set its priority

Lab Task 2

- **Check-off**
  - Demo and explain your results to TA (print out sufficient information to show priority inheritance does happen)
  - Submit your code to Canvas

Thank you!

Have fun with your Raspberry Pi!