


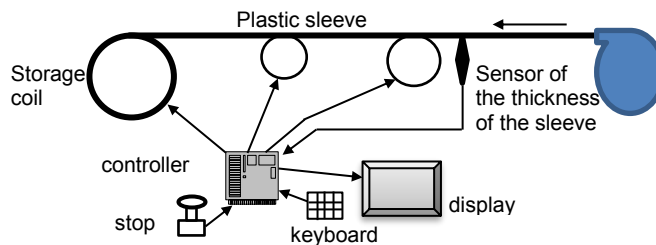



1. a) (1 point) Explain the difference between **interference** and **blocking** in the context of real-time scheduling. 
- b) (1 point) Explain the difference between **periodic**, **sporadic** and **aperiodic tasks** from the point of view of CPU utilization. 
- c) (1 point) Define a **deadline** and the associated typical **taxonomy**. 
2. Consider an industrial machine that produces plastic sleeve (an extruder) composed of several cylinders that extract the sleeve and store it in a coil. To produce the sleeve with a desired constant thickness it is necessary to keep the cylinders rotating with constant speed and control the speed of the storage coil to produce a constant traction on the sleeve.





To control the speeds of the cylinders and coil the machine uses a microcontroller that executes a **preemptive deadline-driven real-time operating system** (uses an EDF scheduler). The control application is composed of the following tasks:



(time units in *ms*)

- $\tau_1 \rightarrow C_1=2, D_1=5, T_1=10$ (periodic, control the speed of the coil)
- $\tau_2 \rightarrow C_2=1, D_2=T_2=10$ (periodic, control the speed of the cylinders)
- $\tau_3 \rightarrow C_3=1, D_3=3, T_3=10000$ (sporadic, stop)
- $\tau_4 \rightarrow C_4=2, D_4=T_4=20$ (sporadic, keyboard)
- $\tau_5 \rightarrow C_5=20, D_5=T_5=100$ (periodic, display)

The controller has an operator interface based on a keyboard and a display. There is, also, an emergency stop button that, once pressed, stops the machine. 

- a) (2 points) Can schedulability be guaranteed with a CPU utilization test? 
- b) (3 points) Test the system schedulability using the CPU demand approach. 
- c) (1 point) Plot the Gantt chart of the tasks execution during the synchronous busy interval (if you did not compute the synchronous busy interval in the previous question, plot the chart from $t=0$ to $t=40ms$). 
- d) (1 point) Consider that the two control tasks and the keyboard task communicate through a shared structure that indicates control gains and setpoints. Reading / writing on it takes 0,5ms. 
Say whether an access synchronization method is needed, if yes, which would you suggest, and which blocking times could each task suffer.

(--- answer the following questions in a separate sheet ---)

3. A computer running a Real-Time POSIX operating system (OS) supporting pre-emptive scheduling is used to control a 3 axis machine. This computer is configured to run three POSIX processes, with each process dedicated to controlling one of the axis. These three processes execute the same binary/executable program (named *axis_ctrl*), while the identification of the axis to be controlled is passed as a command line parameter (e.g. “\$axis_ctrl /dev/axis0”).

The *axis_ctrl* program launches two threads with distinct priorities, each using the `SCHED_RR` and `SCHED_FIFO` scheduling policies. Notice that the system will end up with 6 runnable threads, as the main thread of the program will simply wait for the other threads to finish.









The OS has the round robin time quantum set to 25ms. In POSIX, higher priority numbers correspond to higher priorities.

```
int buf[10];
```

```
//periodic, T=50ms, C=5ms, D=T
void *task1(pthread_mutex_t mut *mutex)
{ t1_init();
  struct sched_param sp;
  sp.sched_priority = 50;
  sched_setscheduler(0, SCHED_FIFO, &sp);
  while(1) {
    t1_run(buf1); // do work (5 ms)
    t1_sleep(); // wait for next period
  } // while(1)
}
```

```
//periodic, T=200ms, C=35ms, D=T
void *task2(pthread_mutex_t mut *mutex)
{ t2_init();
  struct sched_param sp;
  sp.sched_priority = 40;
  sched_setscheduler(0, SCHED_RR, &sp);
  while(1) {
    t2_run(buf2); // do work (35ms)
    t2_sleep(); // wait for next period
  } // while(1)
}
```

```
int main(int argc, char **argv)
{ pthread_attr_t ta;
  pthread_t tid1, tid2;
  pthread_mutex_t mut;
  pthread_mutex_init(&mut, NULL);
  pthread_attr_init(&ta);
  mlockall();
  process_args(argc, argv);
  pthread_create(&tid1, &ta, task1, &mut);
  pthread_create(&tid2, &ta, task2, &mut);
  pthread_join(tid1, NULL);
  pthread_join(tid2, NULL);
  return 0;
}
```

- a) (1,5 points) Once compiled, the final executable program assumes that the code is always copied to the same initial memory address before execution may start. Explain how three distinct processes running the exact same code may co-exist, if each process needs to use the same memory addresses. 
- b) (1,5 points) Assume all 6 tasks/threads are released simultaneously at a point in time we will call (t=0), and no other tasks exist. Draw the sequence of execution of all 6 threads from t=0 up to t=200ms. 
- c) (1 point) “This program will only have correct timely execution if the operating system uses system wide contention for thread scheduling.” Do you agree? Why? 
- d) (1 point) Threads task1() and task2() both need to access a common buffer (buf[10], declared as a global variable). This access is protected by a mutex shared between the threads. Explain why this mutex should be configured to use a priority inheritance / ceiling protocol? Which of the two protocols (PTHREAD_PRIO_INHERIT or PTHREAD_PRIO_PROTECT) would you choose? Explain your reasoning. 
- e) (1,5 points) Notice that the mutex variable (pthread_mutex_t mut) is declared inside the main() function as a local variable.
- Would declaring the mutex as a global variable affect where the variable is located in the process' memory? Explain. 
 - Would declaring the mutex as a global variable result in a single mutex being shared between all 6 threads, and therefore provide an incorrect synchronisation? Why/why not? 
4. (2 points) Describe the distinct phases/steps a compiler' goes through (from source code to the final executable code) while compiling a C program that is broken up into multiple C files and respective headers, as well as using standard libraries. Clearly explain which files are involved, and their respective roles, as well as all the intermediate results. 
5. (1,5 points) In the context of highly dependable systems, explain the meaning of the words 'failure', 'fault', and 'error' 

END

FORMULAE

Rate-Monotonic least upper bound	$U(n) = \sum_{i=1}^n (C_i/T_i) \leq n(2^{1/n}-1)$
Rate-Monotonic hyperbolic bound	$\prod_{i=1}^n (C_i/T_i + 1) \leq 2$
Fixed-priorities response Time	$R_{wc_i}(0) = C_i + B_i + \sum_{k \in hp(i)} C_k$ $R_{wc_i}(m+1) = C_i + B_i + \sum_{k \in hp(i)} \lceil R_{wc_i}(m)/T_k \rceil * C_k$
Synchronous busy period	$L(0) = \sum_i C_i$ $L(m+1) = \sum_i \lceil L(m)/T_i \rceil * C_i$
CPU demand load function	$h(t) = \sum_{D_i \leq t} (1 + \lfloor (t - D_i)/T_i \rfloor) * C_i$