Real Time Systems (TTK4147) – Ex2: Concurrency

Kjetil Kjeka, kjetibk@stud.ntnu.no

Norwegian University of Science and Technology

Today:

- Processes and threads
- Semaphores and mutexes
- Deadlock
- Dining philosophers problem

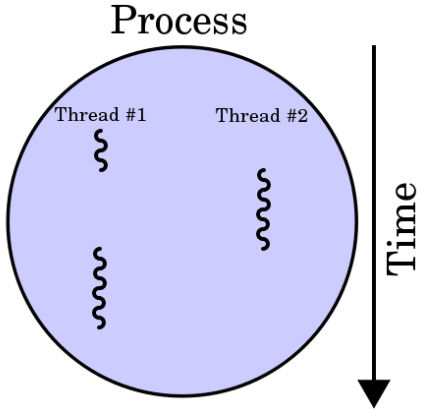
Processes

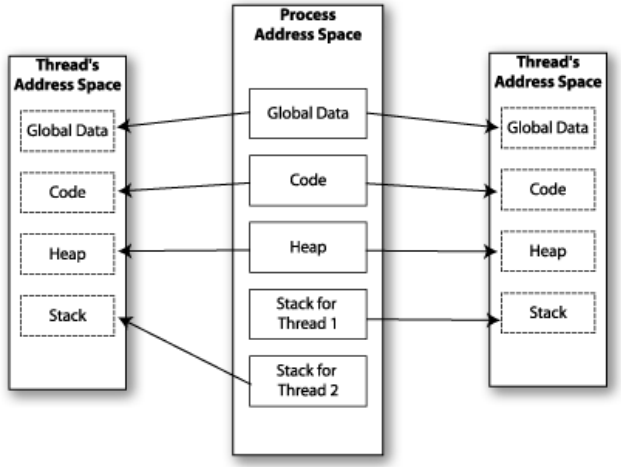
- Executing instance of a program
- C-program executable: `./your_program`
- `fork()` vs. `vfork()`

```
pid_t pid = fork();
if (pid == 0){
     // child process
}
else if( pid > 0){
     // parent process process
}
else{
     // fork failed
}
```

Threads

- Exists within a process
- Exist within the process address space but have their own private stacks






```
static void *thread_fn(void *data){
     /*Do something*/
     pthread_exit(NULL);
}

void main(void){
     pthread_t thread;
     void *data;
     pthread_create(&thread, NULL, thread_fn, (void *) data);
     pthread_join(thread, NULL);
}
```

Semaphores

- Semaphore is an integer count
  - Used to coordinate access to resources
  - Need not be acquired and released by the same thread
- Types:
  - Counting semaphore: Semaphores which allow arbitrary resource count
  - Binary semaphore: 0 or 1

```
sem_t mySem;
if( sem_init(&mySem,1,1) < 0)
{
     perror("semaphore initilization");
     exit(0);
}
sem_wait(&mySem);
     // do something
sem_post(&mySem);
```

Mutexes

- Binary semaphores supporting the concept of ownership
- Can only be released by the thread that locked it

```
pthread_mutex_t lock;
if (pthread_mutex_init(&lock, NULL) != 0)
{
     perror("mutex initilization");
     exit(0);
}
pthread_mutex_lock(&lock);
     // Critical section
pthread_mutex_unlock(&lock);
```

Deadlock

- Two or more threads are blocking each other
- Example:
  - Thread A has locked mutex 1, wants to lock mutex 2
  - Thread B has locked mutex 2, wants to lock mutex 1

Dining philosophers problem

- Classic concurrency problem
- Tip: Solve on paper first