Concurrency and Processes of Pthreads
Pthreads

- Pthreads is a POSIX standard for describing a thread model, it specifies the API and the semantics of the calls.
  - POSIX: Portable Operating System Interface of UNIX
- Model popular – nowadays practically all major thread libraries on Unix systems are Pthreads-compatible
  - Solaris, FreeBSD, Linux
  - Pthreads-win32
Preliminaries

- Include `pthread.h` in the main file
- Compile program with `-lpthread`
  - `gcc -pthread -o test test.c`
  - may not report compilation errors otherwise but calls will fail
- Good idea to check return values on common functions
Thread basic API

- **Types:** `pthread_t` – type of a thread
- **Some calls:**

```c
int pthread_create(pthread_t *thread,
                   const pthread_attr_t *attr,
                   void * (*start_routine)(void *),
                   void *arg);

int pthread_join(pthread_t thread, void **status);
int pthread_detach();
void pthread_exit();
```

- No explicit parent/child model, except main thread holds process info
- Call `pthread_exit` in main, don’t just fall through;
- Most likely you wouldn’t need `pthread_join`
  - `status` = exit value returned by joinable thread
- Detached threads are those which cannot be joined (can also set this at creation)
```c
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS 5

void* PrintHello(void *threadid){
    printf("\n%d: Hello World!\n", threadid);
    pthread_exit(NULL);
}

int main (int argc, char *argv[]){
    pthread_t threads[NUM_THREADS];
    int args[NUM_THREADS];
    int rc, t;
    for(t=0;t < NUM_THREADS;t++){
        printf("Creating thread %d\n", t);
        args[t] = t;
        rc = pthread_create(&threads[t], NULL, PrintHello,
                          (void *) args[t]);

        if (rc) {
            printf("ERROR: return code from pthread_create() is %d\n", rc);
            exit(-1);
        }
    }
    pthread_exit(NULL);
}
```
Attributes

• **Type:** `pthread_attr_t` *(see `pthread_create`)*

• Attributes define the state of the new thread

• Attributes: system scope, joinable, stack size, inheritance... you can use default behaviors with `NULL` in `pthread_create`

```c
int pthread_attr_init(pthread_attr_t *attr);
int pthread_attr_destroy(pthread_attr_t *attr);
pthread_attr_{set/get}{attribute}
```

• **Example:**

```c
pthread_attr_t attr;
pthread_attr_init(&attr); // Needed!!!
pthread_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
pthread_attr_setscope(&attr, PTHREAD_SCOPE_SYSTEM);
pthread_create(NULL, &attr, foo, NULL);
```
Pthread Mutexes

- **Type:** `pthread_mutex_t`

```c
int pthread_mutex_init(pthread_mutex_t *mutex,
                       const pthread_mutexattr_t *attr);
int pthread_mutex_destroy(pthread_mutex_t *mutex);
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
int pthread_mutex_trylock(pthread_mutex_t *mutex);
```

- **Attributes:** for shared mutexes/condition vars among processes, for priority inheritance, etc.
  - use defaults

- **Important:** Mutex scope must be visible to all threads!
Pthread semaphore

- int sem_init(sem_t *sem, int pshared, unsigned int value);
- int sem_wait(sem_t *sem);
- int sem_post(sem_t *sem);
The Dining Philosophers Problem

• Philosophers
  – think
  – take forks (one at a time)
  – eat
  – put forks (one at a time)

• Eating requires 2 forks
• Pick one fork at a time
• How to prevent deadlock?
• What about starvation?
• What about concurrency?
Dining philosophers: definition

- Each process needs two resources
- Every pair of processes compete for a specific resource
- A *process may proceed only if it is assigned both resources*
- Every process that is waiting for a resource should sleep (be blocked)
- Every process that releases its two resources must wake-up the two competing processes for these resources, if they are interested
Dining Philosophers Problem
An incorrect naïve solution

(→ means “waiting for this fork”)

Echtzeitsysteme

WS 16/17

Lehrstuhl Informatik VI – Robotics and Embedded Systems
The LR Solution

- If the philosopher acquires one fork and the other fork is not immediately available, she holds the acquired fork until the other fork is free.

- Two types of philosophers:
  - L -- The philosopher first obtains its left fork and then its right fork.
  - R -- The philosopher first obtains its right fork and then its left fork.

- The LR solution: the philosophers are assigned acquisition strategies as follows: philosopher \( i \) is R-type if \( i \) is even, L-type if \( i \) is odd.
Assumption: “the fork is fair”.

(→ means “first fork taken”)
Thank you!
I6Lehre W 16