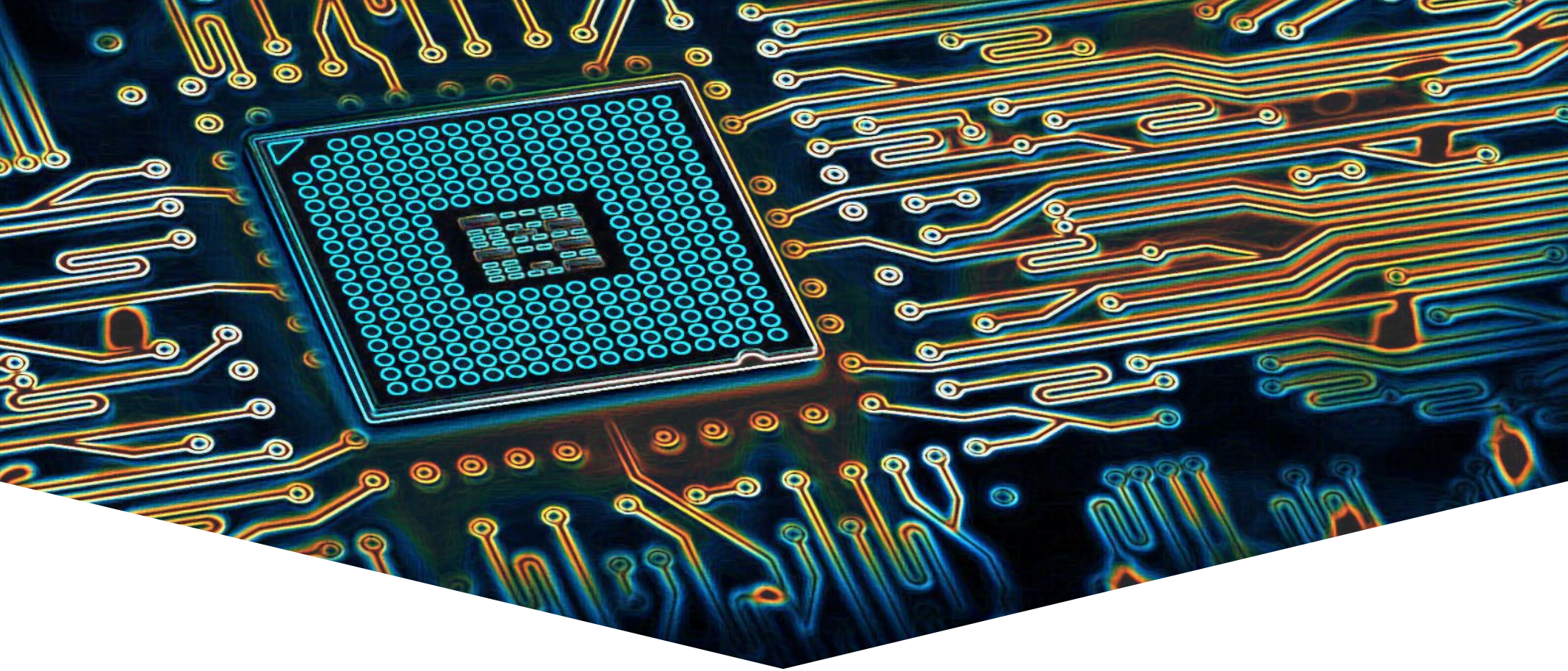




RTS1 Week 5

- Tasks
 - Creation / Deletion
 - Parameter passing
- Multitasking problems
 - Situation / Problem
 - Solution

“An environment where program execution can be interrupted and continued at any time in any location”

Questions

- How to design such a system and promise timing?
- How to prevent data corruption?
- How to communicate between tasks?



IEEE

THE *Open* GROUP

POSIX

- Portable Operating System Interface (POSIX) is a standard **API** for Operating Systems.
 - Many OS partially comply with this standard. For example: Linux, Android, OSX, VxWorks, QNX Neutrino, **free-RTOS** etc.
- Tasks (**threads**) are dynamically created by using API calls.
- **Semaphores** and **mutexes** can be used to synchronize tasks.
- **Message Queues** can be used to communicate between tasks.

Pthread Example (1 of 2)

```
void *print1(void *par) {  
    for (int i = 0; i < 10; i++) {  
        usleep(100000);  
        printf("print1\n");  
    }  
    return NULL;  
}
```

```
void *print2(void *par) {  
    for (int i = 0; i < 10; i++) {  
        usleep(200000);  
        printf("print2\n");  
    }  
    return NULL;  
}
```

Pthread Example (2 of 2)

REAL-TIME SYSTEMS

Uitvoer: print1
print2
print1
print1
print2
print1
print1
print2
print1
print1
print2
print1
print1
print2
print2
print2
print2
print2
print2

```
void *main_thread(void *arg) {  
    pthread_attr_t pta;  
    pthread_attr_init(&pta);  
    pthread_attr_setstacksize(&pta, 1024);  
  
    pthread_t t1, t2;  
    pthread_create(&t1, &pta, &print1, NULL);  
    pthread_create(&t2, &pta, &print2, NULL);  
  
    pthread_join(t1, NULL);  
    pthread_join(t2, NULL);  
  
    pthread_attr_destroy(&pta);  
  
    return NULL;  
}
```

Source: pthread.c

Pthread with Parameter Example (1 of 2)

REAL-TIME SYSTEMS

```
typedef struct {  
    char *msg;  
    useconds_t us;  
} par_t;  
  
void *print(void *par) {  
    par_t* p = par;  
    for (int i = 0; i < 10; i++) {  
        usleep(p->us);  
        printf(p->msg);  
    }  
    return NULL;  
}
```

Pthread with Parameter Example (2 of 2)

REAL-TIME SYSTEMS

Uitvoer: print1
print2
print1
print1
print2
print1
print1
print2
print1
print1
print2
print1
print1
print2
print2
print2
print2
print2
print2

```
void *main_thread(void *arg) {  
    pthread_attr_t pta;  
    pthread_attr_init(&pta);  
    pthread_attr_setstacksize(&pta, 1024);  
  
    pthread_t t1, t2;  
    par_t p1 = {"print1\n", 100000};  
    par_t p2 = {"print2\n", 200000};  
    pthread_create(&t1, &pta, &print, &p1);  
    pthread_create(&t2, &pta, &print, &p2);  
  
    pthread_join(t1, NULL);  
    pthread_join(t2, NULL);  
    pthread_attr_destroy(&pta);  
  
    return NULL;  
}
```

Source: [pthread_par.c](#)

Problem with Shared Memory

REAL-TIME SYSTEMS

```
volatile int aantal = 0;
```

Source: [pthread_shared.c](#)

```
void *teller(void *par) {  
    for (int i = 0; i < 100000; i++) {  
        aantal++;  
    }  
    return NULL;  
}
```

```
//...
```

```
pthread_create(&t1, &pta, &teller, NULL);  
pthread_create(&t2, &pta, &teller, NULL);  
pthread_create(&t3, &pta, &teller, NULL);
```

What is the final
value of aantal?

Problem with Shared Memory

- The operation `aantal++` is **not atomic** (in machine code).
- For example, R4 contains the address of `aantal`:

```
LDR.N R0, [R4, #0]  
ADD.N R0, R0, #1  
STR.N R0, [R4, #0]
```

What happens when
a task switch occurs
at this moment?

- What is the minimal and the maximal final value of `aantal`?
- Minimum = 1000000
- Maximum = 3000000

Data Corruption

Situations: Task A and B use a shared global variable
(just demonstrated)

Task C and D are both using the same peripheral
(e.g., UART port)

Goal: Preventing concurrent use of a resource by multiple tasks

Solution: Using tokens to represent resources. Allow a limited number of tasks to get the same token at the same time.

Solution?

- There are solutions which use shared variables (2 flags and 1 turn variable) and **busy waiting**.
- Dekker's algorithm: http://en.wikipedia.org/wiki/Dekker's_algorithm
- Peterson's algorithm: http://en.wikipedia.org/wiki/Peterson's_algorithm
- Busy waiting **costs** clock cycles!
- OSes offer solutions **without** busy waiting.

IPC Inter Process (Task) Communication

REAL-TIME SYSTEMS

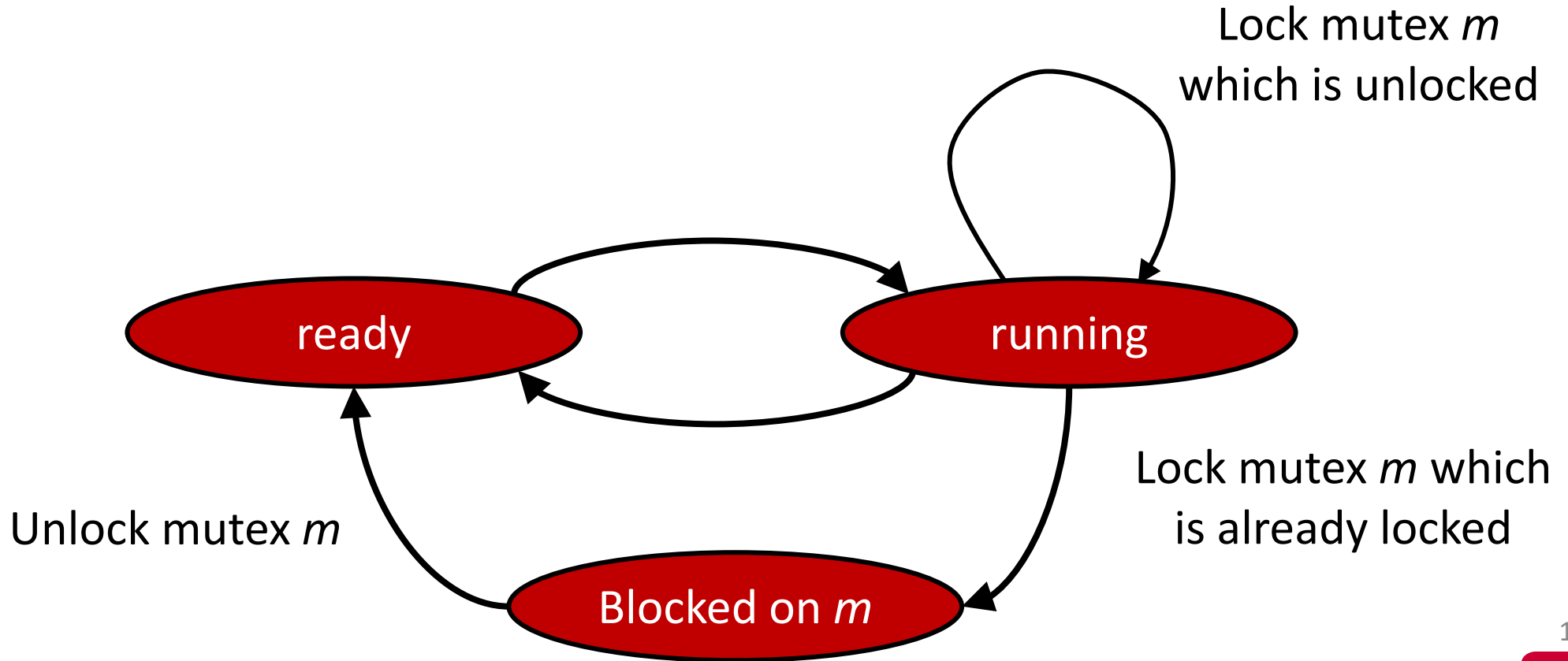
- **Shared variable** based
 - Dekker's or Peterson's algorithm
 - Busy waiting (inefficient)
 - Spinlock
 - Busy waiting (inefficient)
 - **Mutex**
 - **Semaphore**
 - Monitor
 - Mutex combined with Conditional variables
 - Barrier
 - Read Write Lock
 - Event Groups
- **Message** based
 - **Message Queue**

- Simple way to create a **mutual exclusive** so-called critical section.
- Only **one** task can be in the critical section.
- Mutex has a **lock** (take) and a **unlock** (give) function.
 - OS ensures that these functions are **atomic**!
 - At the **start** of the critical section the mutex must be **locked** (taken) and at the **end** of the critical section the mutex must be **unlocked** (given).



Task States

REAL-TIME SYSTEMS



- When a task t tries to lock mutex m which is already locked by another task, **task t is blocked on m .**
We also say:
 - Task t **waits for** mutex m .
 - Task t **sleeps until** mutex m is unlocked.
- **Order** of unblocking (waking up):
 - general purpose OS: FIFO
 - real-time OS: highest **priority**



Mutex with Shared Memory

```
int aantal = 0;
pthread_mutex_t m;

void *teller(void *par) {
    for (int i = 0; i < 100000; i++) {
        pthread_mutex_lock(&m);
        aantal++;
        pthread_mutex_unlock(&m);
    }
    return NULL;
}
```

Source: mutex.c

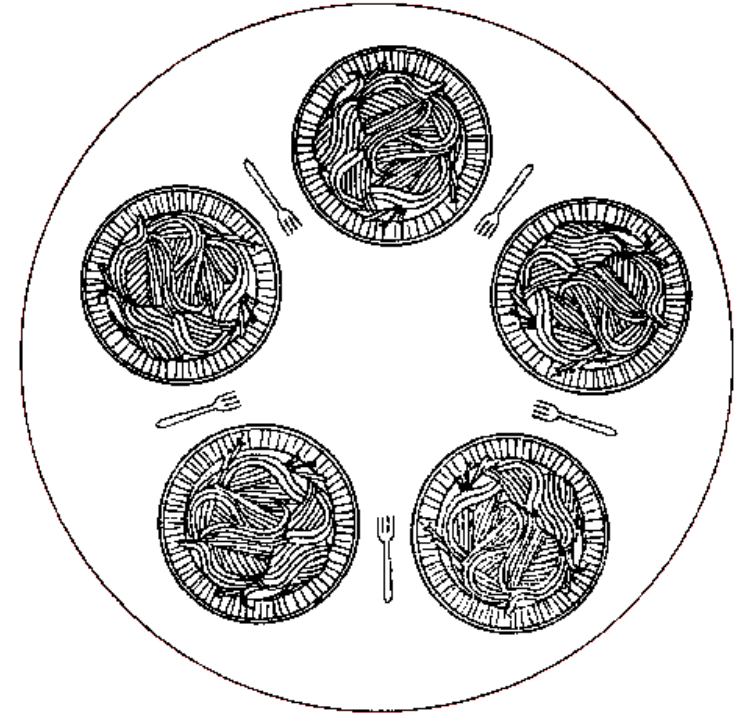
DANGER

- Priority inversion
 - Low priority task has mutex locked
 - High priority task is blocked due to mutex
 - Solution: **priority inheritance**
- Deadlock
 - Task A has resource 1 locked and wants to lock resource 2
 - Task B has resource 2 locked and wants to lock resource 1

Will be discussed in week 6!

Deadlock (Example)

- There are 5 philosophers.
- The life of a philosopher consists of:
 - Thinking;
 - Eating.
- Each philosopher has one plate and one fork.
- To eat a philosopher needs two forks.



Dining Philosophers

```
pthread_mutex_t dinner_fork[5];
void *philosopher(void *par) {
    int i = *(int*)par;
    while (1) {
        printf("philosopher %d is sleeping\n", i);
        usleep(100000 * (rand() % 5 + 1));
        printf("philosopher %d (tries to) picks up left fork\n", i);
        pthread_mutex_lock(&dinner_fork[i]);
        printf("philosopher %d (tries to) picks up right fork\n", i);
        pthread_mutex_lock(&dinner_fork[(i + 1) % 5]);
        printf("philosopher %d is eating\n", i);
        usleep(100000 * (rand() % 5 + 1));
        pthread_mutex_unlock(&dinner_fork[i]);
        printf("philosopher %d lies down left fork\n", i);
        pthread_mutex_unlock(&dinner_fork[(i + 1) % 5]);
        printf("philosopher %d lies down right fork\n", i);
    }
    return NULL;
}
```

Dining Philosophers

```
pthread_t t[5];
int tid[5];
for (int i = 0; i < 5; i++)
{
    tid[i] = i;
    pthread_create(&t[i], &pta, &philosopher, &tid[i]);
}
```

This program can come to a hold (deadlock)! How?

Solution? See homework assignment hereafter.

Source: [pthread_philosophers.c](#)

Counting Semaphore

- Operations:
 - **Psem** (prolaag (probeer te verlagen), take, **wait**):
wait (block, sleep) if count ≤ 0 else decrement count.
 - **Vsem** (verhoog, signal, give, **post**):
unblock a waiting task if count ≤ 0 else increment count.
- Order of unblocking (wake up):
 - general purpose OS: **FIFO**
 - real-time OS: highest **priority**



Edsger Dijkstra

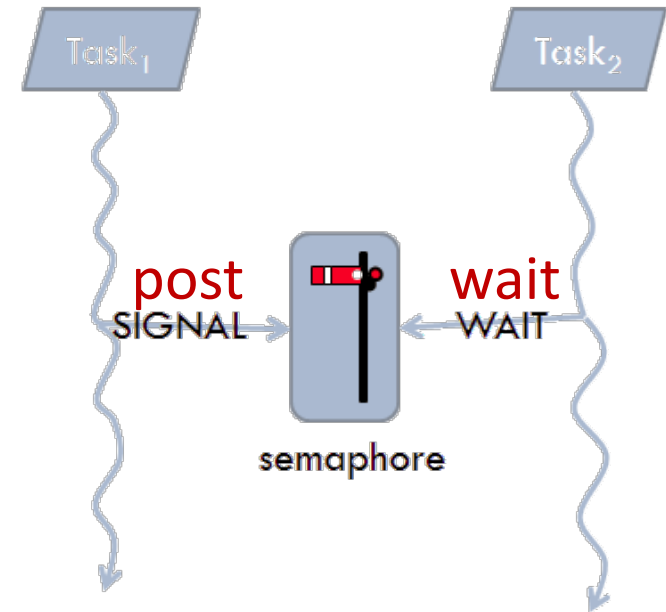
- Solve the problem of the dining philosophers by using a **semaphore** to ensure that no more than 4 philosophers will be able to sit at the table at the same time.

Solution: Create semaphore with initial count of 4 in main. Now each philosopher must take a semaphore after sleeping and must give the semaphore before sleeping again.

The semaphore counts the number of available chairs at the table and a philosopher requires a chair to sit at the table.

Semaphore versus Mutex

- **Mutex** can only be used for **mutual exclusion** (task which takes the mutex should also give the mutex (back)).
- **Semaphore** can also be used for other **synchronization** purposes.
- Homework:
 - Task a consists of two sequential parts a1 and a2.
 - Task b consists of two sequential parts b1 and b2.
 - Task c consists of two sequential parts c1 and c2.
 - Make sure (using a semaphore) that the parts b2 and c2 are always executed **after** part a1 has been executed.



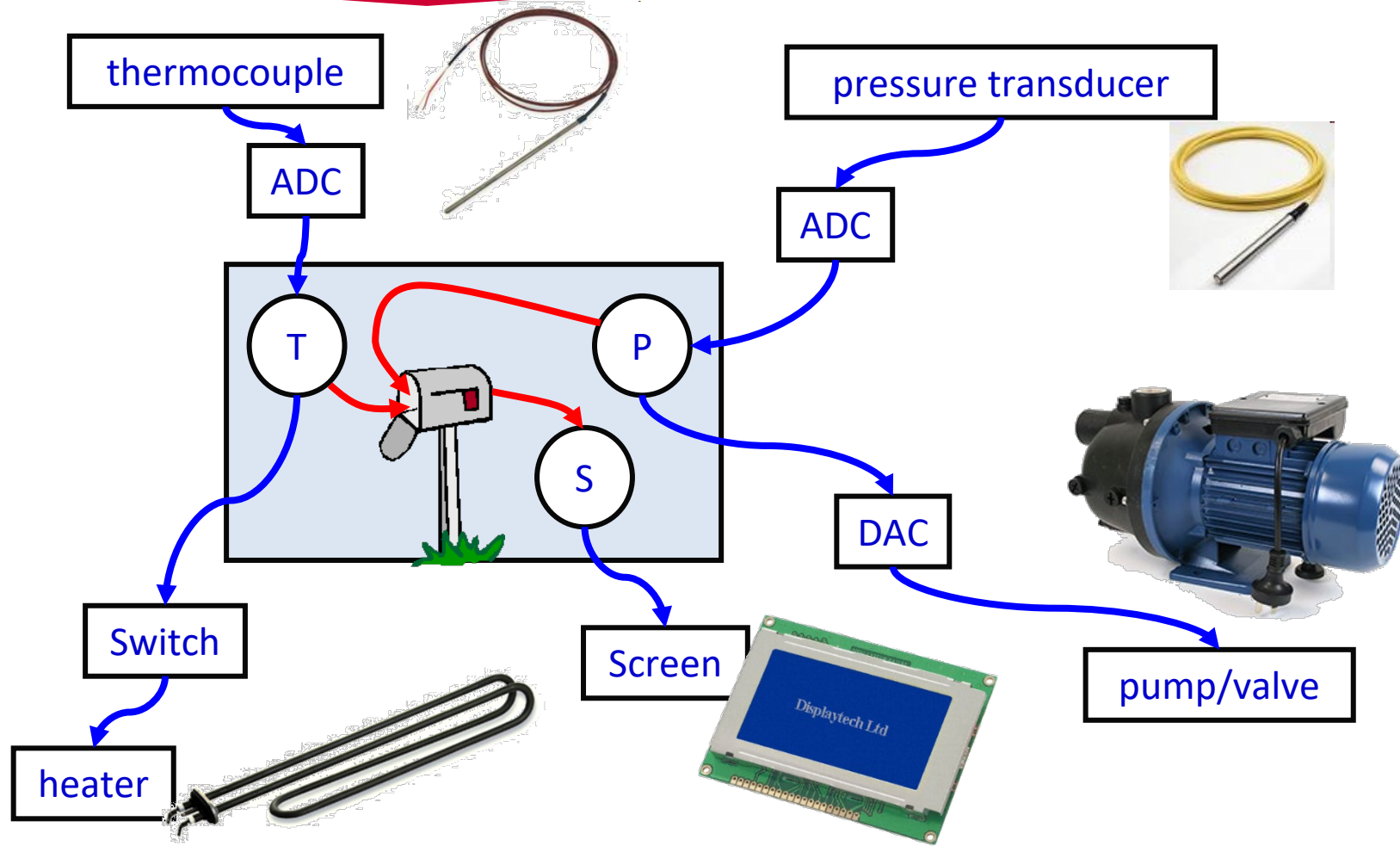
IPC Inter Process (Task) Communication

REAL-TIME SYSTEMS

- **Shared variable** based
 - Dekker's or Peterson's algorithm
 - Busy waiting (inefficient)
 - Spinlock
 - Busy waiting (inefficient)
 - **Mutex**
 - **Semaphore**
 - Monitor
 - Mutex combined with Conditional variables
 - Barrier
 - Read Write Lock
 - Event Groups
- **Message** based
 - **Message Queue**

Communication between Threads

REAL-TIME SYSTEMS



Message Queue Example (1 of 2)

```
void *main_thread(void *arg) {
    mqd_t mqdes;
    struct mq_attr mqAttrs;
    mqAttrs.mq_maxmsg = 3;
    mqAttrs.mq_msgsize = sizeof(intb;
    mqAttrs.mq_flags = 0;
    mqdes = mq_open("/ints", O_RDWR | O_CREAT, 0666, &mqAttrs);

    pthread_t tp, tc;
    pthread_attr_t attr;
    pthread_attr_init(&attr);
    pthread_attr_setstacksize(&attr, 1024);
    pthread_create(&tp, &attr, &producer, &mqdes);
    pthread_create(&tc, &attr, &consumer, &mqdes);
}
```

Message Queue Example (2 of 2)

```
void *producer(void *p) {
    mqd_t mq = *(mqd_t *)p;
    for (int i = 0; i < 10; i++) {
        mq_send(mq, (char *)&i, sizeof(i), 0);
    }
    return NULL;
}
```

```
void *consumer(void *p) {
    mqd_t mq = *(mqd_t *)p;
    for (int i = 0; i < 10; i++) {
        int msg;
        mq_receive(mq, (char *)&msg, sizeof(msg), NULL);
        printf("%d\n", msg);
    }
    return NULL;
}
```

Source: [mqueue.c](#)

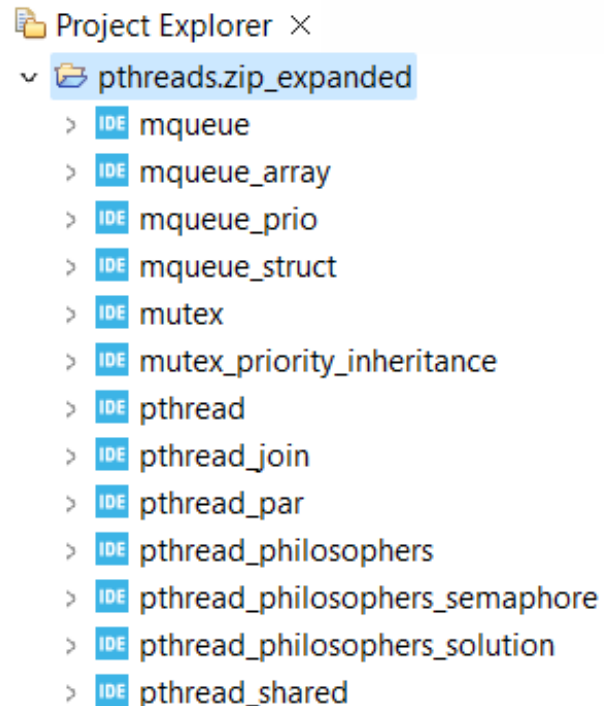
- Er zijn kennisclips over bovenstaande onderwerpen te vinden op:
<https://hrelektrotechniek.bitbucket.io/pthread/>
- Daarin wordt de CC3220S van TI gebruikt i.p.v. de STM32F411E-DISCO maar dat maakt niet zo veel uit.



Example projects

All example projects can be found at:

<https://github.com/HR-ELEKTRO/rts1/wiki/pthreads>



Next week ...

REAL-TIME SYSTEMS

Schedulability en response time analyses

Aan de slag!

REAL-TIME SYSTEMS

Aan de slag met [Opdrachten_Week_5.pdf](#)

