HOGESCHOOL ROTTERDAM

Knowledge Clip

Embedded Systems

pthread
mutex

brojz@hr.nl

Problem with Shared Memory

Source: [pthread_shared.c]

```
volatile int aantal = 0;

void *teller(void *par) {
    for (int i = 0; i < 10000000; i++) {
        aantal++;
    }
    return NULL;
}

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

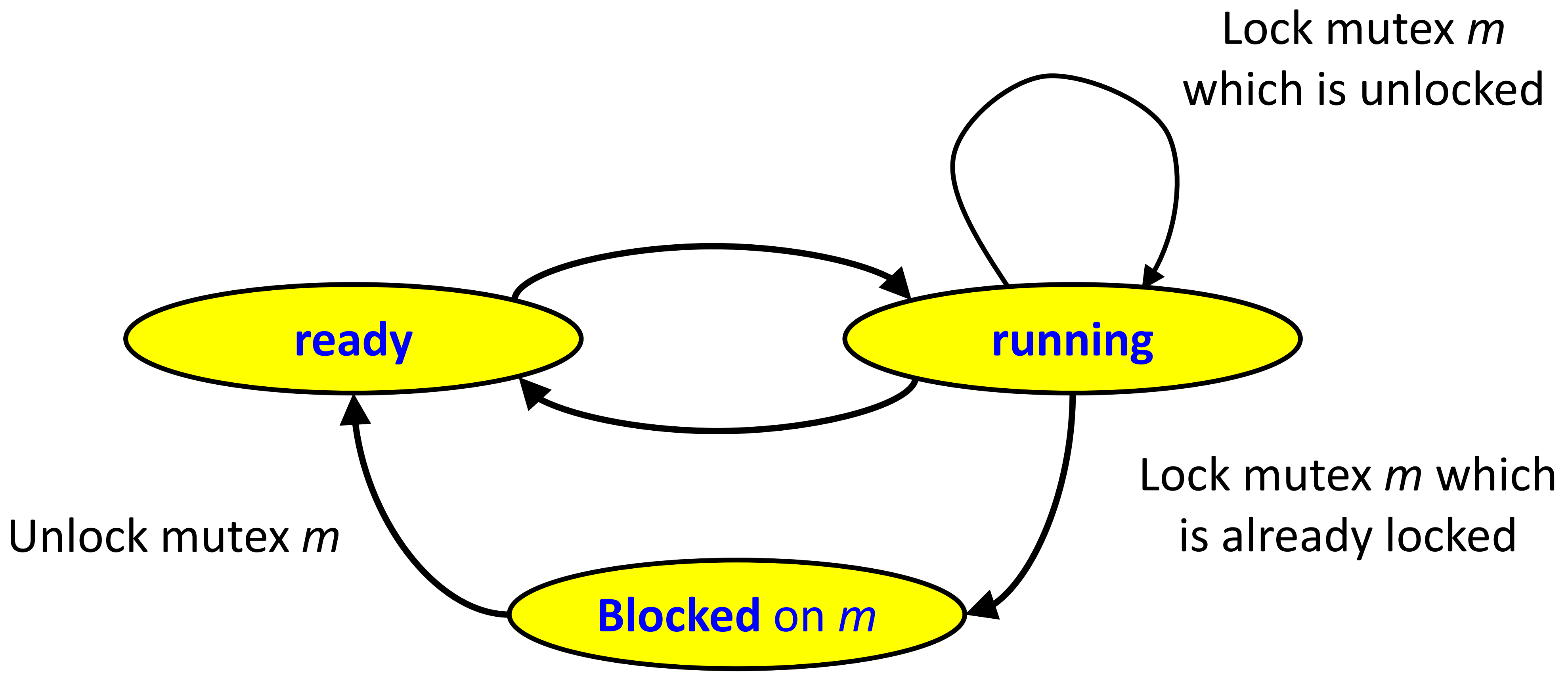
What is the final value of aantal?

Mutex

- 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


Lock mutex *m* which is unlocked
ready
running
Unlock mutex *m*
Lock mutex *m* which is already locked
Blocked on *m*

exceed expectations

HOGESCHOOL ROTTERDAM

Mutex

- 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

exceed expectations
HOGESCHOOL ROTTERDAM

Mutex with Shared Memory

```
int aantal = 0;
pthread_mutex_t m;

void *teller(void *par) {
    for (int i = 0; i < 10000000; i++) {
        pthread_mutex_lock(&m);
        aantal++;
        pthread_mutex_unlock(&m);
    }
    return NULL;
}
```

Source: mutex.c

Danger

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