CPSC 457 Tutorial
Week 4

System calls

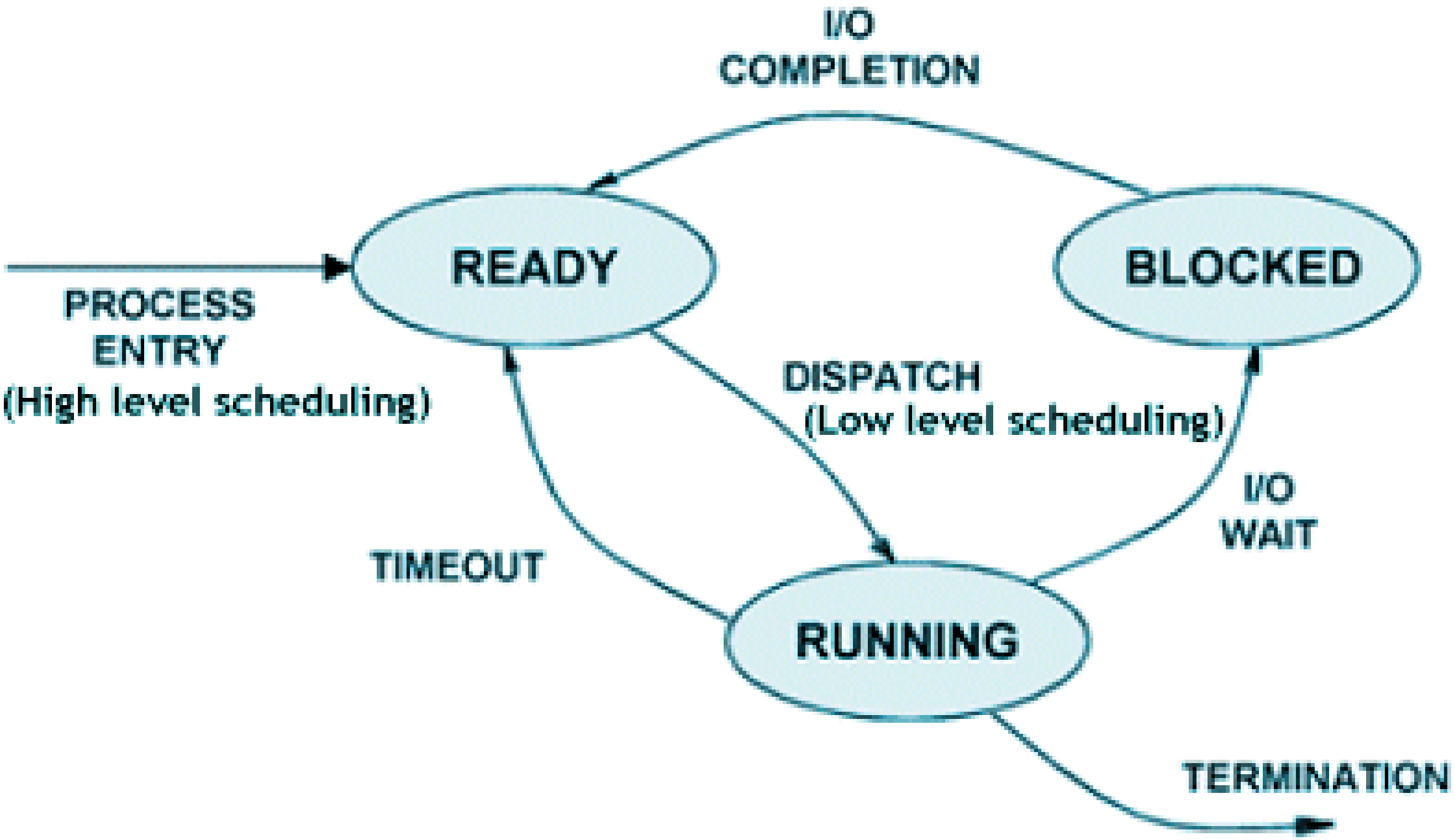

I/O COMPLETION
READY
BLOCKED
PROCESS ENTRY (High level scheduling)
DISPATCH (Low level scheduling)
I/O WAIT
TIMEOUT
RUNNING
TERMINATION

Figure 1: Process state diagram

Source: https://www.technologyuk.net/computing/operating-systems/process-management.shtml

Threads

- Basic unit of CPU utilization

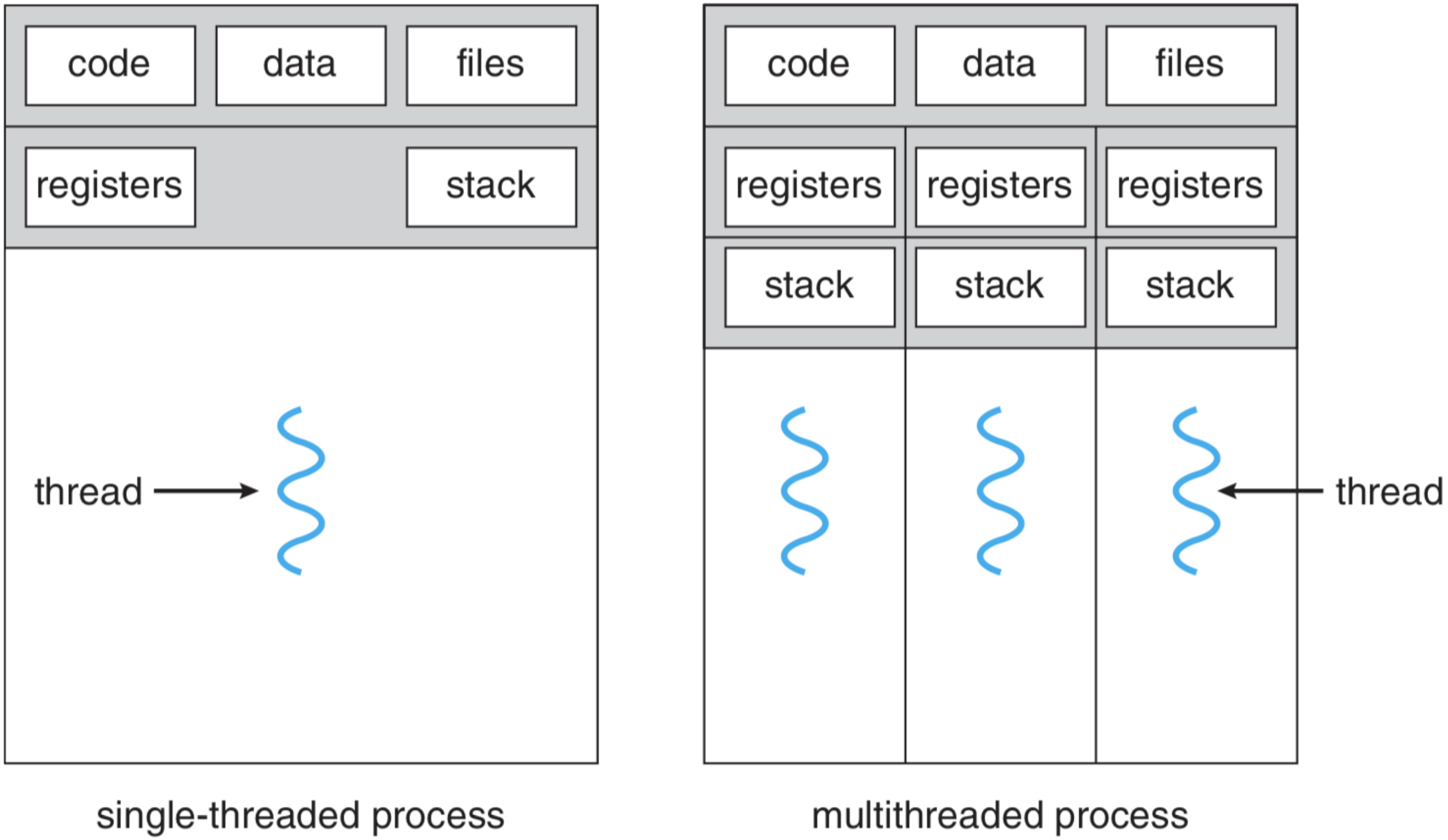


Figure 2: Single-threaded and multithreaded processes

Taken from Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. 2012. Operating System Concepts (9th ed.)

Threads (cont'd)

- Imagine an IDE program
- There will be multiple jobs performed simultaneously(!) while you are coding
  - Auto updater
  - Spell checker
  - Auto compilation
  - Text editor

Benefits

- **Responsiveness:** When one of the tasks is blocked, the others can continue
- **Resource sharing:** Sharing code and data allows running multiple threads on the same address space
- **Performance:** It is faster to create threads as they share resources of the process
- **Scalability:** While a single threaded process runs on one core, multiple threads may be running on different cores

POSIX threads

- man pthreads contains descriptions and examples

- int **pthread_create**(pthread_t **thread*, const pthread_attr_t **attr*, void *(**start_routine*) (void *), void **arg*)

- void **pthread_exit**(void **value_ptr*)

- int **pthread_join**(pthread_t *thread*, void ***value_ptr*);

Thread concurrency

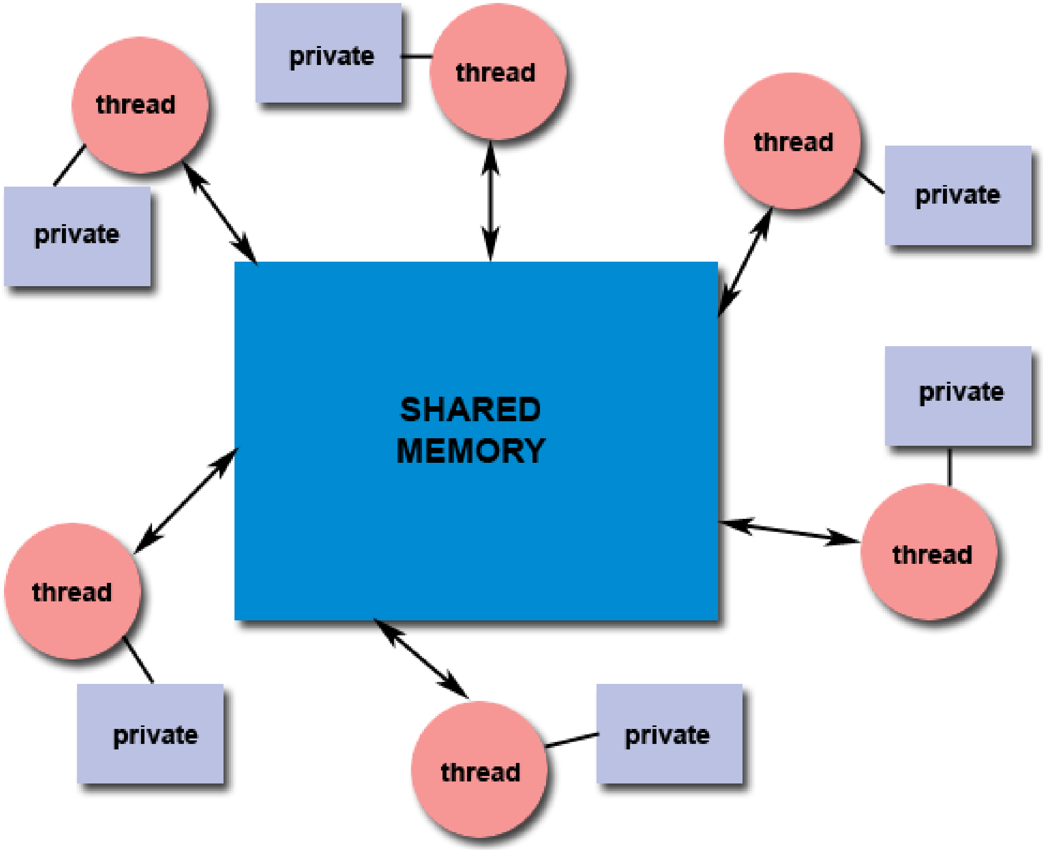
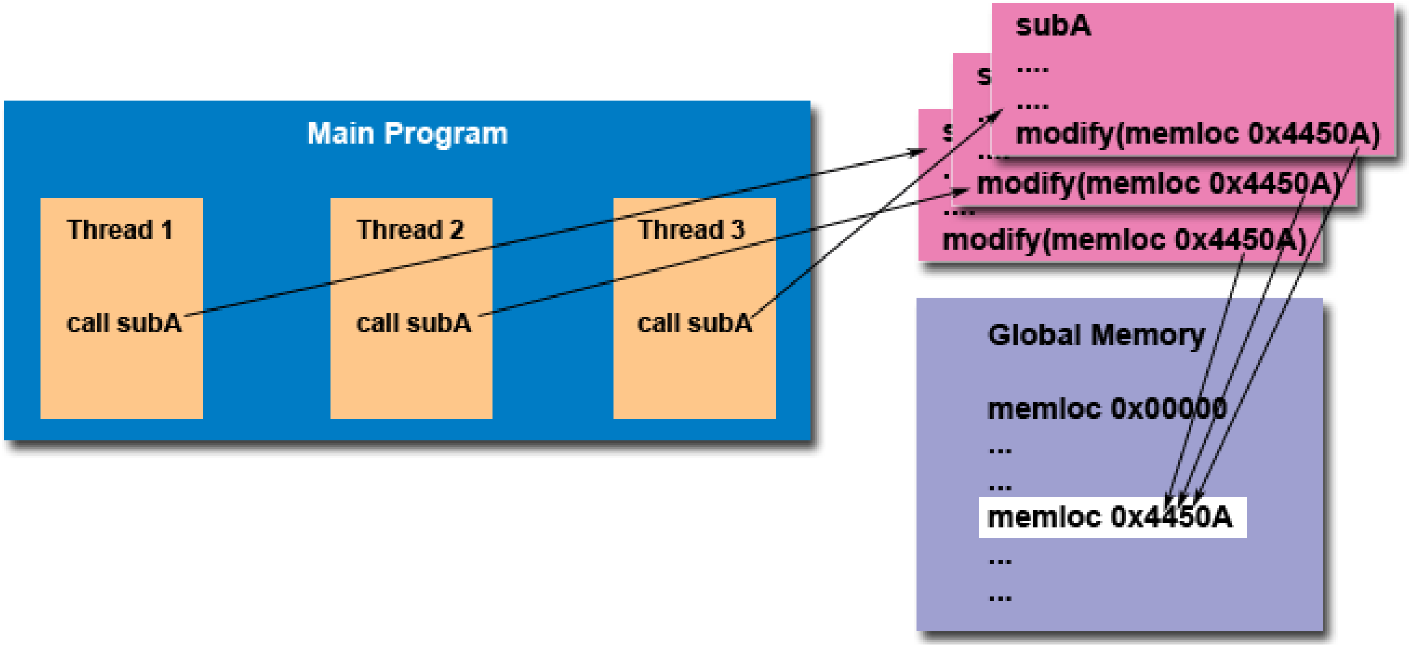


Figure 1: Multithread model

Taken from https://computing.llnl.gov/tutorials/pthreads/

Thread concurrency (cont'd)

- Accessing and modifying shared data should be secured in multithreaded processes
- Otherwise **race conditions** may occur in **critical sections**



Taken from https://computing.llnl.gov/tutorials/pthreads/

Race condition example

```
int cnt = 0;

void* incr(void* args) {
    cnt++;
    return NULL;
}
```

load cnt to reg0
increment reg0
store reg0 back to cnt

Thread 1	Thread 2	cnt
reg0 = cnt		0
reg0 = reg0 + 1		0
	reg1 = cnt	0
	reg1 = reg1 + 1	0
cnt = reg0		1
	cnt = reg1	1

Mutex

- Short of *mutual exclusion*
- Used to prevent race conditions
- There are two main operations:
  - lock()
  - unlock()
- These operations are atomic, meaning that only one thread can take the lock
- When a mutex is taken by one of the threads, the others wait for it to be released

POSIX Mutex

- pthread_mutex_t for type
- pthread_mutex_init(pthread_mutex_t *)
- pthread_mutex_lock(pthread_mutex_t *)
- pthread_mutex_unlock(pthread_mutex_t *)
- pthread_mutex_destroy(pthread_mutex_t *)