

Operating Systems

Lab. Class

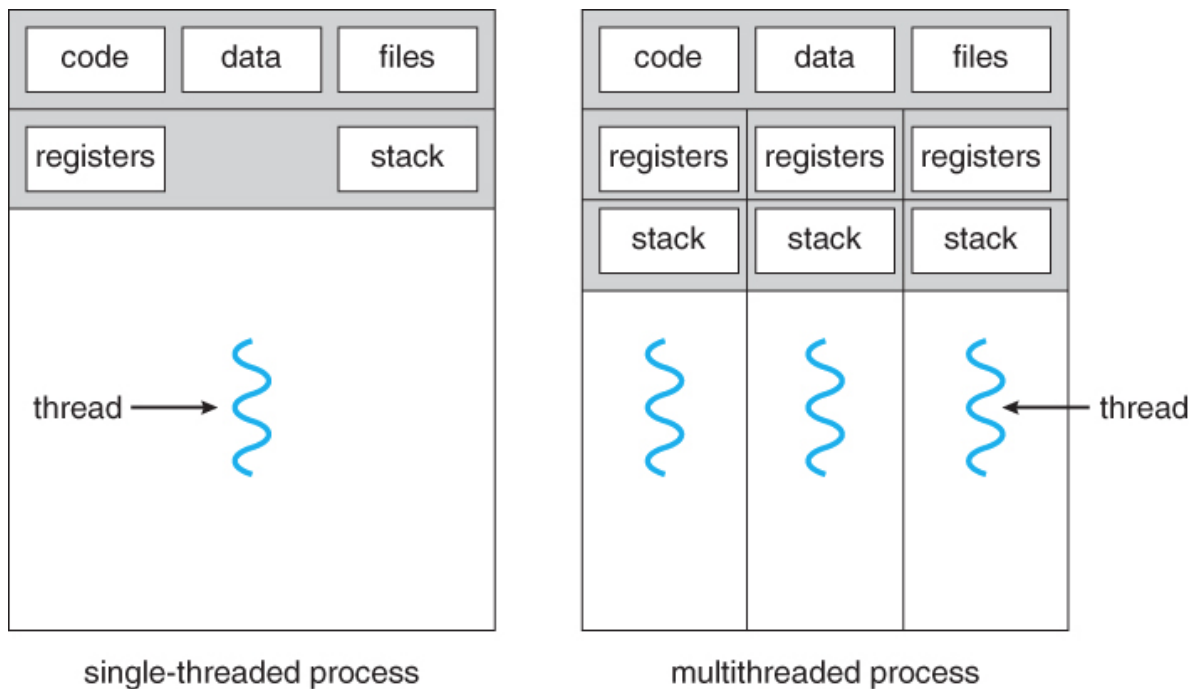
Week 11

Project Plan

- 6 projects
 0. Install xv6
 1. System call
 2. Scheduling
 3. Virtual memory 1
 4. Virtual memory 2
 5. Concurrency 1
 6. Concurrency 2
- Individual projects

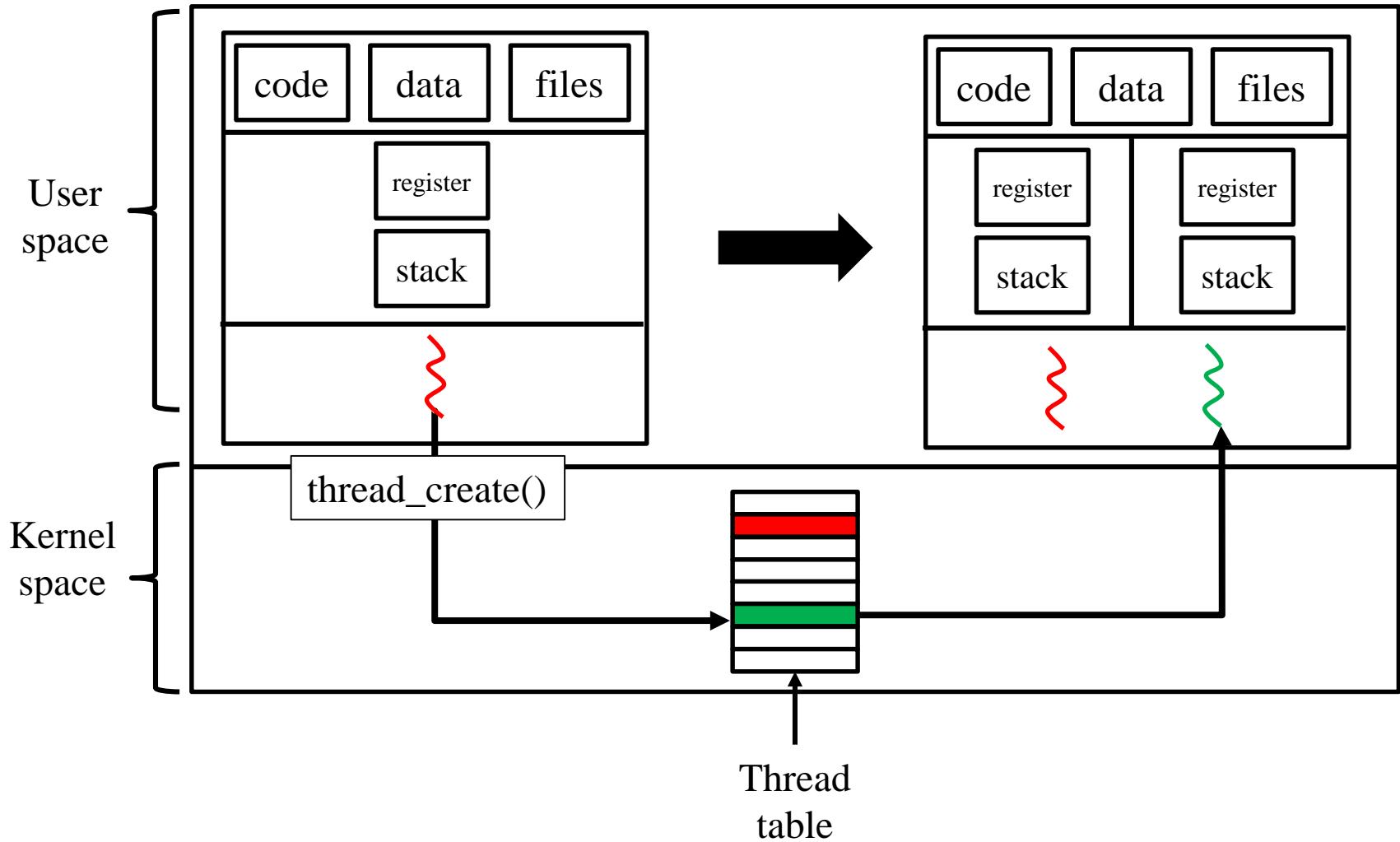
Supporting Threads on Xv6

- The original xv6 process is **single-threaded**.
- A multithreaded process consists of one or more threads.
 - Each thread has its own call stack.
 - Every thread shares code, data, and other resources such as open files



https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/4_Threads.html

Supporting Threads on Xv6 (Cont'd)



Supporting Threads on Xv6 – thread_create()

Name

thread_create – create a new thread

Synopsis

```
int thread_create(void *(*function)(void *), int priority, void *arg, void *stack);
```

Description

The thread_create() starts a new thread in the calling process. The new thread starts execution by invoking function(); arg is passed as the sole argument of function(). priority is the scheduling priority of the new thread (0~40). stack is the pointer to call stack of new thread.

Return value

Return the thread ID (tid) of the new thread. tid is guaranteed to be unique within a process. On error, return -1.

Supporting Threads on Xv6 – thread_exit()

Name

thread_exit – terminate calling thread

Synopsis

```
void thread_exit(void *retval);
```

Description

The thread_exit() function terminates the calling thread and returns a value via `retval` that is available to another thread in the same process that calls thread_join().

Return value

This function does not return to the caller.

Supporting Threads on Xv6 – thread_join()

Name

thread_join – join with a terminated thread

Synopsis

```
int thread_join(int tid, void **retval);
```

Description

The thread_join() function waits for the thread specified by tid to terminate. If that thread has already terminated, then thread_join() returns immediately.

thread_join() copies the exit status of the target thread into the location pointed to by *retval. The call stack of the terminated thread should be freed by the calling thread.

Return value

On success, return 0. If there's no thread with input tid, return -1.

Supporting Threads on Xv6 – gettid()

Name

gettid() – get thread identification

Synopsis

```
int gettid(void);
```

Description

The `gettid()` function returns caller's thread ID (TID). If the process is a single-threaded process, the thread ID is same as the process ID. In a multi-thread process, all threads have the same PID, but each one has a unique TID within a process.

Return value

Return the thread ID of calling thread.

Supporting Threads on Xv6 – getpid()

Name

getpid() – get process identification

Synopsis

```
int getpid(void);
```

Description

The `getpid()` function returns the caller's process ID (PID). On multi-threaded process, every thread of the same process returns same PID.

Return value

Return the process ID of calling process.

PA #5 – Thread support on Xv6

- Implement the following system calls on xv6
 - thread_create()
 - thread_exit()
 - thread_join()
 - gettid()
- Modify the following system call to support threads
 - getpid()
- Implement a priority scheduler which supports threads

- Submission deadline
 - 2016-05-29 23:59

PA #5 – Things to Consider (1)

- We assume that each thread always terminates by calling `thread_exit()`.
- If the main thread terminates or any thread calls `exit()`, the whole process is terminated. In this case, all the threads should be terminated as well. Also, address space should be freed and open files should be closed.
- Open files are shared among threads. If thread A opens a file, the file can be also accessed by another thread B (in the same process) using the same file descriptor. Files opened by thread A need not be closed automatically when thread A terminates.

PA #5 – Things to Consider (2)

- When a thread calls `thread_exit()`, the thread remains in the zombie state until another thread calls `thread_join()`.
- There is no parent-child hierarchy among threads. Any thread can invoke `thread_join()` for another thread.
- All threads within a process should return the same PID. Thread IDs are guaranteed to be unique only within a process.
- The maximum number of threads per process is limited to 8 (including the main thread). Your implementation should support at least 32 processes.

PA #5 Template Code

- Download from <http://sys.skku.edu>
- Modifications
 - Remove debug messages
 - Do not print any messages on screen
 - halt system call
 - Halt xv6 program
 - New system calls for supporting threads (in thread.c)
 - thread_create(), thread_exit(), thread_join(), gettid()
 - Existing system call for supporting threads (in proc.c)
 - getpid()
 - Multithreaded test program
 - threadtest.c
 - make tarball
 - Compress your source codes into one .tar.gz file for submission
 - You should enter your ID & project no. on Makefile

Project Submission Procedure

- <http://sys.skku.edu>
- Since 2nd submission, adds -5% penalty of the project score
- Every one day delay, -25% penalty of the project score
 - You can use up to 5 *slip* days

PA #5 Test Cases

- Test cases will be uploaded to <http://csl.skku.edu/SWE3004S16/Projects>
- Our thread interface is a simplified version of POSIX Pthreads interface. Refer to manual pages on `pthread_create()`, `pthread_join()`, and `pthread_exit()`.