IOWA STATE UNIVERSITY
Department of Electrical and Computer Engineering

Lecture 08: Threads

Agenda

- Recap
- Threads
  - Thread Concept
  - POSIX Threads

Recap

- Process Creation: fork()

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]){
    printf("hello world (pid:%d)\n", (int) getpid()); //get process ID
    int rc = fork();       // create a child process
    if (rc < 0) {          // fork failed; exit
        fprintf(stderr, "fork failed\n");
        exit(1);
    } else if (rc == 0) { // child (new process)
        printf("hello, I am child (pid:%d)\n", (int) getpid());
    } else {               // parent goes down this path (main)
        printf("hello, I am parent of %d (pid:%d)\n",
        rc, (int) getpid());
    }
    return 0;
}
```

Recap

- Other process APIs
  - wait()
    - wait for child process
  - exec()
    - load a new program in child process

- "Zombie" process
  - A process that has completed execution but still has an entry in the process table
  - consume resource if not "reaped"

Agenda

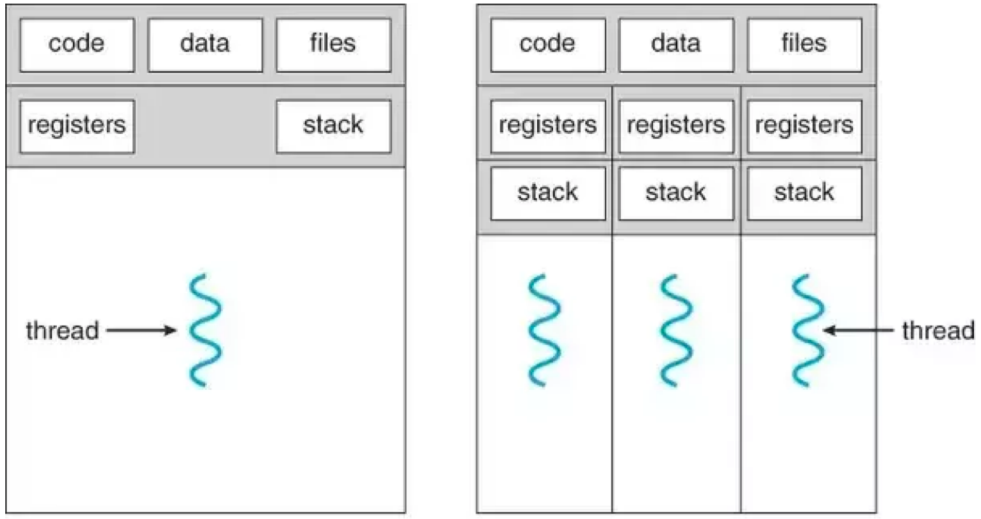
- ~~Recap~~
- Threads
  - Thread Concept
  - POSIX Threads

Threads

- Traditional single-threaded program only has one “thread of control”
  - one program counter (PC)
- Multi-threaded program
  - Multiple threads of control *within* a process
    - All threads within a process share the same
      - text, heap, static data segments, open files ...
    - each thread has its own
      - State, PC, registers, stack

Threads

- Single-thread V.S. multi-thread



single-threaded process multithreaded process

Per-process items	**Per-thread items**
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	

Threads

- Why threads?
  - creating a new thread is much cheaper than creating a new process (e.g., 100 times)
  - Switching between two threads also cheaper
  - “lightweight processes”, “mini-processes”

Agenda

- ~~Recap~~
- Threads
  - ~~Thread Concept~~
  - POSIX Threads

POSIX Threads

- Portable Operating System Interface (POSIX)
  - a set of standards specified by IEEE for maintaining compatibility between operating systems
    - includes a set of C API specifications
      - Most Linux system calls implement specific POSIX C API functions and thus make Linux POSIX-compliant
    - includes a thread library
      - **POSIX Threads**, or **Pthreads**
      - specifies behavior of the thread library
      - implementation is up to developers of the library
      - Common in UNIX-like Oses (Solaris, Linux, Mac OS X)
    - …

Thread Creation

- Create a new thread

```
#include <pthread.h>

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

  - `pthread.h`: header for all pthread functions
  - `thread`: pointer to this thread; used to interact with the thread later
  - `attr`: Used to specify any attributes this thread might have
    - Stack size, Scheduling priority, …
    - can use `NULL` to specify default values
  - `start_routine`: the function this thread start running in.
  - `arg`: the argument to be passed to the function (`start_routine`)
    - *a void pointer* allows us to pass in *any type of* argument.

Thread Creation Example

```
typedef struct __myarg_t {
        int a;
        int b;
} myarg_t;

void *mythread(void *arg) {
        myarg_t *m = (myarg_t *) arg;
        printf("%d %d\n", m->a, m->b);
        return NULL;
}

int main(int argc, char *argv[]) {
        pthread_t p;
        int rc;

        myarg_t args;
        args.a = 10;
        args.b = 20;
        rc = pthread_create(&p, NULL, mythread, &args);
        …
}
```

Thread Join

- Wait for a thread to complete

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

- `thread`: Specify which thread *to wait for*
- `value_ptr`: A pointer to the return value

Thread Join Example

```
1   #include <stdio.h>
2   #include <pthread.h>
3   #include <assert.h>
4   #include <stdlib.h>
5
6   typedef struct __myarg_t {
7       int a;
8       int b;
9   } myarg_t;
10
11  typedef struct __myret_t {
12      int x;
13      int y;
14  } myret_t;
15
16  void *mythread(void *arg) {
17      myarg_t *m = (myarg_t *) arg;
18      printf("%d %d\n", m->a, m->b);
19      myret_t *r = malloc(sizeof(myret_t));
20      r->x = 1;
21      r->y = 2;
22      return (void *) r;
23  }
```

Thread Join Example (cont.)

```
24  int main(int argc, char *argv[]) {
25      int rc;
26      pthread_t p;
27      myret_t *m;
28
29      myarg_t args;
30      args.a = 10;
31      args.b = 20;
32      pthread_create(&p, NULL, mythread, &args);
33      pthread_join(p, (void **) &m);  // this thread has been
                                        // waiting inside of the
                                        // pthread_join() routine.
34      printf("returned %d %d\n", m->x, m->y);
35      free(m);
36      return 0;
37  }
```

Another Example: Dangerous Code

- Be careful with how values are returned from a thread

```
1   void *mythread(void *arg) {
2       myarg_t *m = (myarg_t *) arg;
3       printf("%d %d\n", m->a, m->b);
4       myret_t r;
5       r.x = 1;
6       r.y = 2;
7       return (void *) &r;
8   }
```

Another Example: Dangerous Code

- Be careful with how values are returned from a thread

```
1   void *mythread(void *arg) {
2       myarg_t *m = (myarg_t *) arg;
3       printf("%d %d\n", m->a, m->b);
4       myret_t r; // ALLOCATED ON STACK!
5       r.x = 1;
6       r.y = 2;
7       return (void *) &r;
8   }
```

  - When the function returns, `r` is automatically de-allocated (i.e., the stack frame is destroyed)
    - pointer to `r` will point to invalid data

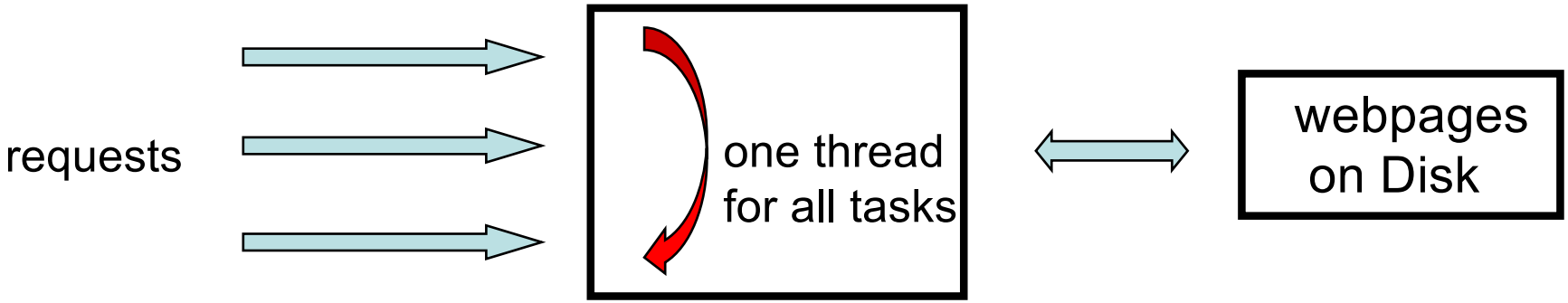
Other Examples

- Creating multiple threads

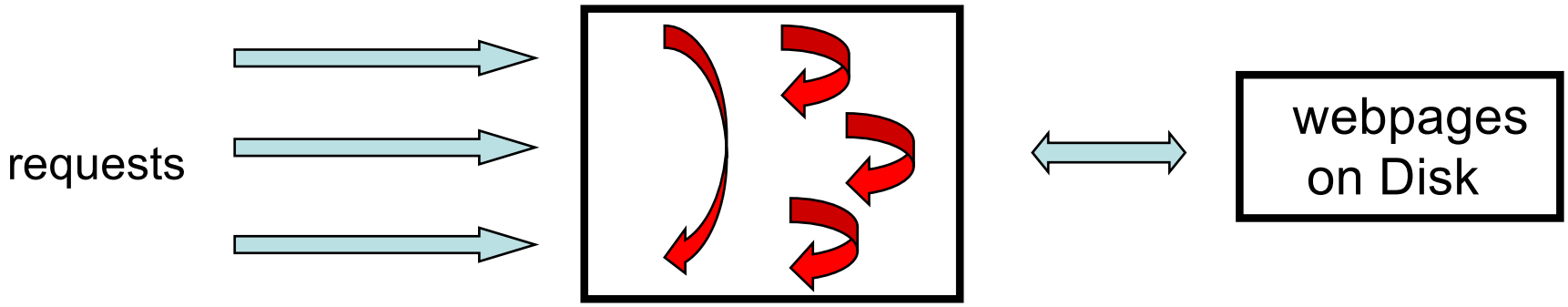
```
void *worker_thread(void *arg) {
        … //do some work based on *arg
}

int main(int argc, char *argv[]) {
        …
        pthread_t threads[N_WORKERS];
        myarg_t myargs[N_WORKERS];
        for (int i=0; i<N_WORKERS; ++i) {

                … //custom myargs[i]

                pthread_create(&threads[i], 0, worker_thread, &myargs[i])
        }
        for (int i=0; i<N_WORKERS; ++i) {
                pthread_join(threads[i], 0);
        }
        …
}
```

Other Examples

- Webserver
  - single-threaded



  - multi-threaded



one dispatcher and multiple workers

Agenda

- ~~Recap~~
- ~~Threads~~
  - ~~Thread Concept~~
  - ~~POSIX Threads~~

Questions?

*acknowledgement: slides include content from “Modern Operating Systems” by A. Tanenbaum, “Operating Systems Concepts” by A. Silberschatz etc., “Operating Systems: Three Easy Pieces” by R. Arpaci-Dusseau etc., and anonymous pictures from internet.