CDA0017: Operating Systems

Donghyun Kang (donghyun@changwon.ac.kr)

NOSLab (https://noslab.github.io)

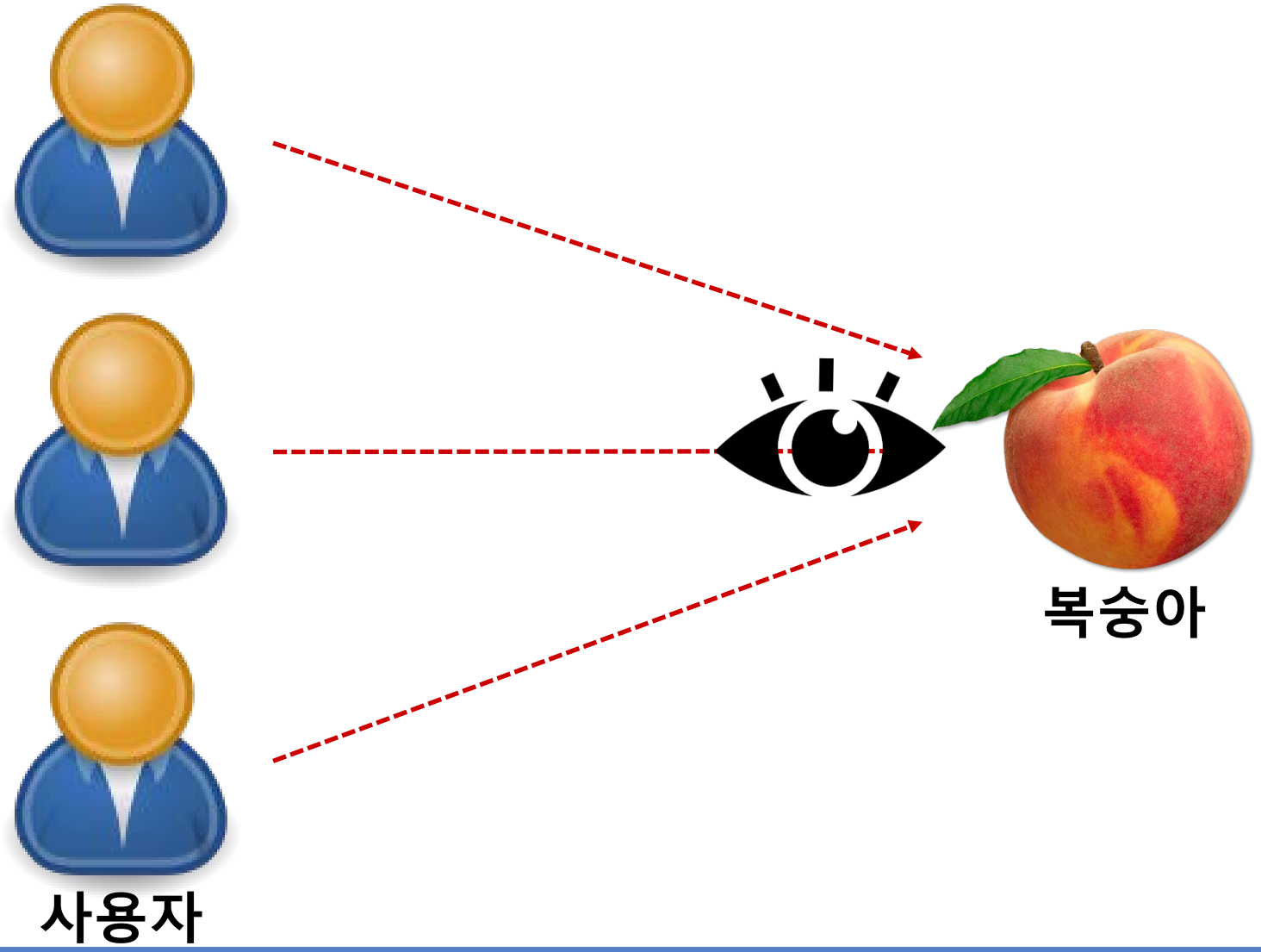
Changwon National University

대화

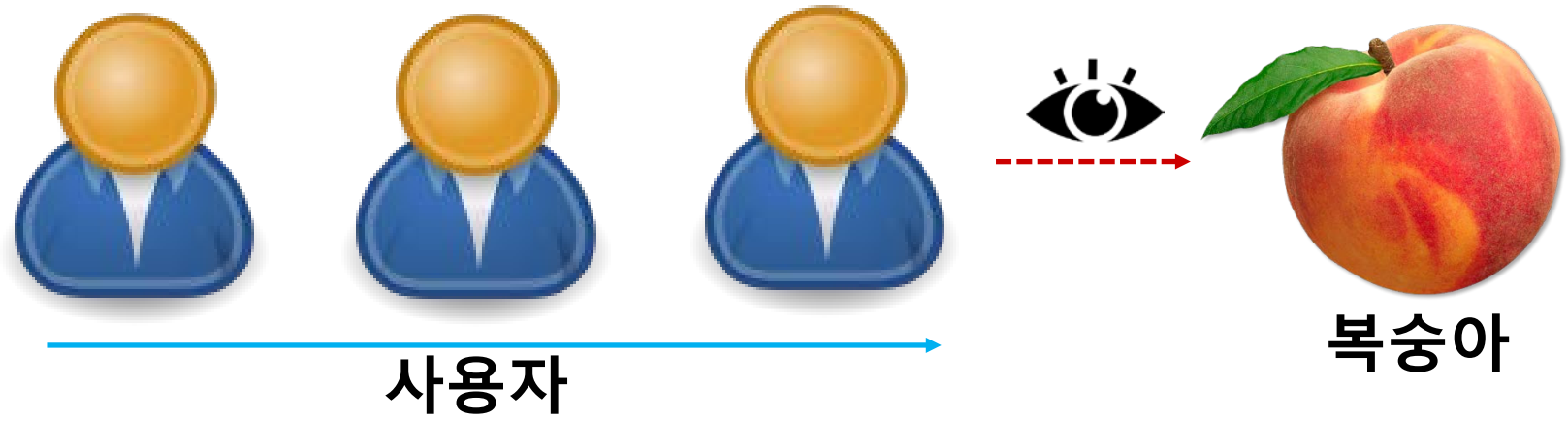
- 교수: 복숭아가 많이 있고 여러 사람이 복숭아를 먹고 싶어 한다고 가정하자. 먹으려는 사람이 먼저 복숭아를 눈으로 확인한 후 복숭아를 집어 먹도록 하면 어떤 문제가 있을까?

- 학생: ??

병행성에 대한 대화



병행성에 대한 대화



병행성의 동기(Motivation)

- 병행성의 이슈
  - 가상 CPU, 가상 메모리
- 프로세스는 프로그램의 추상화 제공
  - 운영체제는 프로셋스간 보호(protection) 및 고립(isolation)을 제공
- 프로세스의 문제점
  - 하나의 프로세스는 멀티 코어를 이용할 수 없음
  - 협업하는 프로세스 기반 프로그램은 작성하기 어려움
    - 프로세스간 통신이 어려움
  - 새로운 프로세스의 생성 비용이 발생함
  - 문맥 교환 비용이 발생함

병행성을 확보하면서 낮은 비용으로 처리할 수 있는 방법은?

쓰레드(Thread)

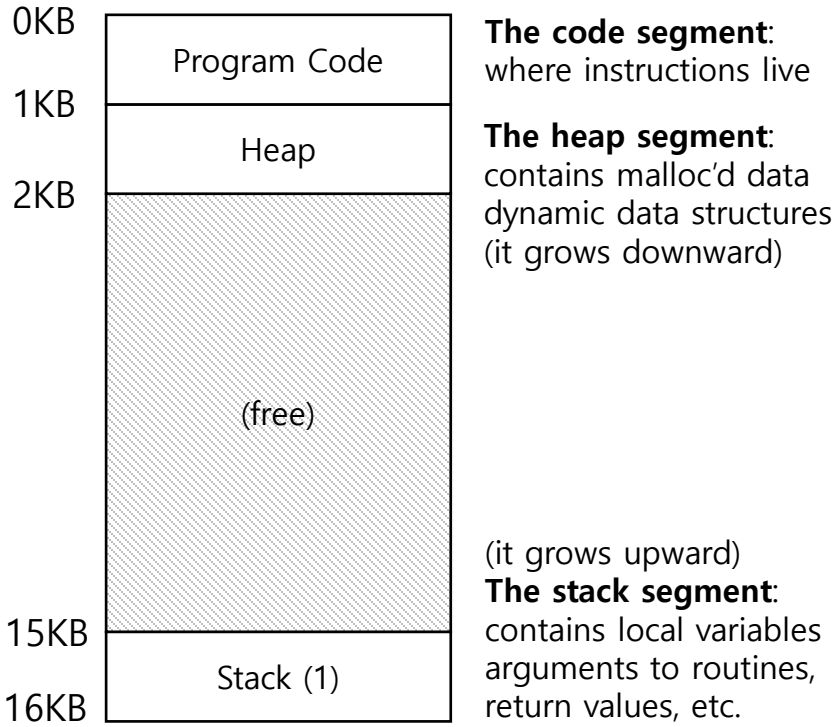
- 프로세스의 새로운 추상화 개념
  - 하나의 프로세스가 하나의 명령어만 실행하지 않고 다수의 명령어를 동시에 실행하도록 제공

- 쓰레드
  - 독립적인 Thread ID
  - 독립적인 Stack
  - 독립적인 PC, SP, 관련 레지스터

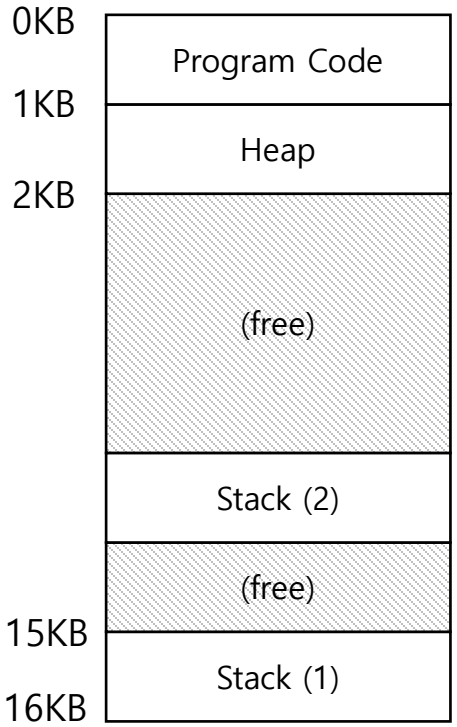
- 멀티-쓰레드 프로그램(Multi-thread program)
  - 동일한 주소 공간(address space)
  - 쓰레드간 문맥 교환 (context switch)
  - 쓰레드 제어 블록(thread control block, TCB)

쓰레드 stack

- 쓰레드 당 하나의 쓰레드 stack 소유



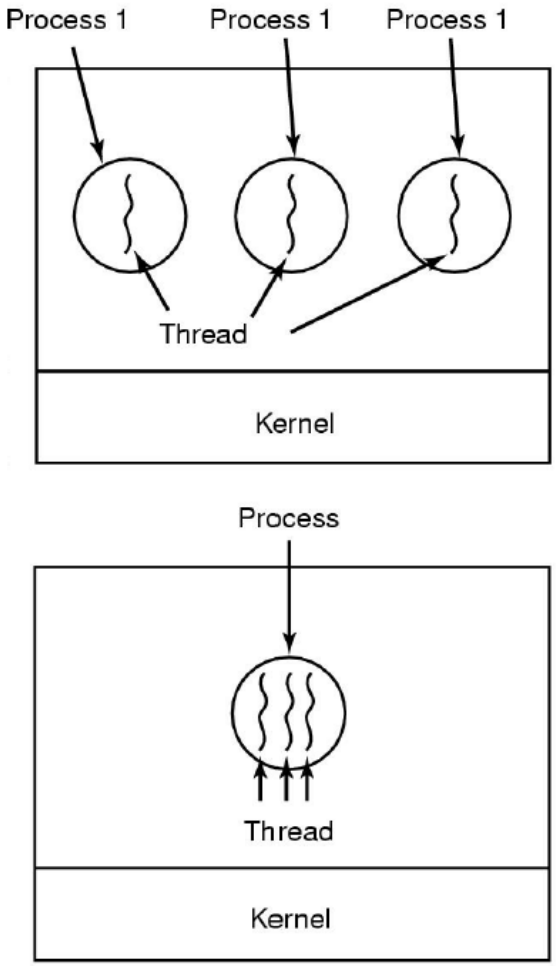
A Single-Threaded Address Space



Two threaded Address Space

쓰레드(Thread)

- Single thread vs. multi-thread



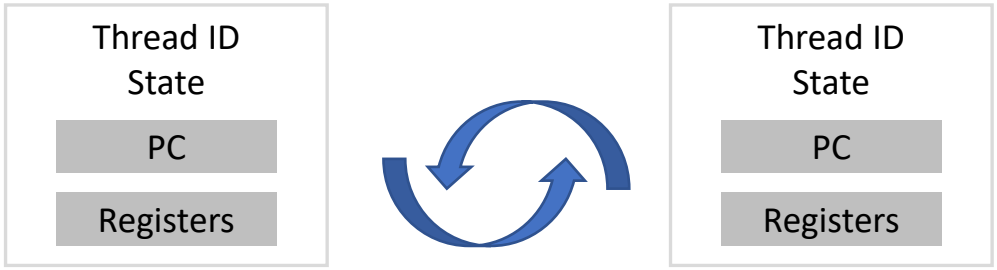
쓰레드 예제

```
#include <stdio.h>
#include <pthread.h>

void *hello (void *arg)
{
    printf (“hello, world\n”);
    ...
}
int main()
{
    pthread_t tid;
    pthread_create (&tid, NULL, hello, NULL);
    printf (“hello from main thread\n”);
    ...
}
```

쓰레드 간 문맥 교환

- 각 쓰레드는 자신만의 program counter와 register들의 집합을 가짐



- 문맥 교환
  - T1의 PC, registers 저장
  - T2의 PC, registers 복원
  - **주소 공간 유지**

문맥 교환 예제

```
1   #include <stdio.h>
2   #include <assert.h>
3   #include <pthread.h>
4
5   void *mythread(void *arg) {
6       printf("%s\n", (char *) arg);
7       return NULL;
8   }
9
10  int
11  main(int argc, char *argv[]) {
12      pthread_t p1, p2;
13      int rc;
14      printf("main: begin\n");
15      rc = pthread_create(&p1, NULL, mythread, "A");
16      assert(rc &=&  0);
17      rc = pthread_create(&p2, NULL, mythread, "B");
18      assert(rc &=&  0);
19      // 종료 할 수 있도록 대기 중인 쓰레드 병합하기
20      rc = pthread_join(p1, NULL); assert(rc &=&  0);
21      rc = pthread_join(p2, NULL); assert(rc &=&  0);
22      printf("main: end\n");
23      return 0;
24  }
```

문맥 교환 예제

Main	쓰레드 1	쓰레드 2
실행 시작		
"main: begin" 출력		
쓰레드 1 생성		
쓰레드 2 생성		
T1을 대기		
	실행	
	"A" 출력	
	리턴	
T2를 대기		
		실행
		"B" 출력
		리턴
"main: end" 출력		

<생성 → 대기 실행>

Main	쓰레드 1	쓰레드 2
실행 시작		
"main: begin" 출력		
쓰레드 1 생성		
	실행	
	"A" 출력	
	리턴	
쓰레드 2 생성		
		실행
		"B" 출력
		리턴
T1을 대기		
즉시 리턴; T1 완료		
T2를 대기		
즉시 리턴; T2 완료		
"main: end" 출력		

<생성 직후 실행>

Processes vs. Threads

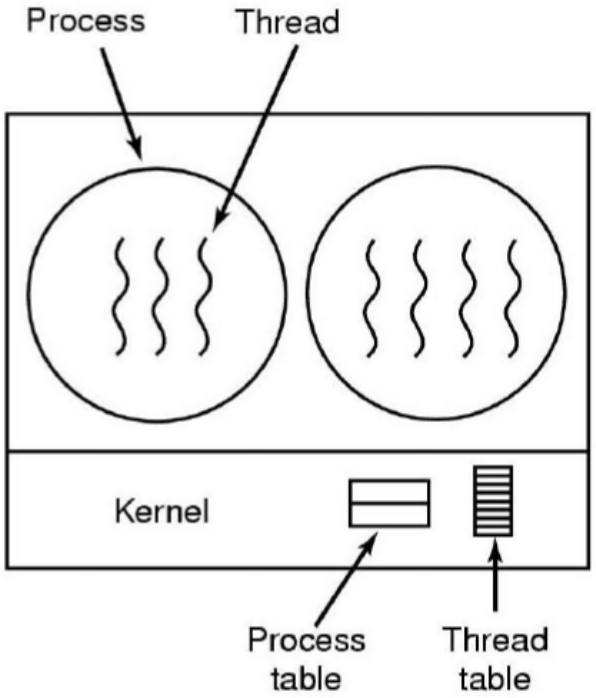
- 하나의 쓰레드는 하나의 프로세스에 종속됨
- 그러나, 하나의 프로세스는 다수의 쓰레드를 가질 수 있음
- 쓰레드는 동일한 주소 공간을 사용하기 때문에 데이터의 공유 비용이 최소화 됨
- 쓰레드는 스케줄링의 최소 단위임
- 프로세스는 쓰레드가 실행되는 컨테이너 역할
  - PID, address space, user and group ID, open file descriptors, current working directory, etc.
- 프로세스는 정적이나, 쓰레드는 동적임

쓰레드 사용의 장점

- 병행성 비용이 저렴함
- 프로그램의 구조를 향상시킬 수 있음
  - 협력 쓰레드를 통해서 작은 작업 단위로 분할 가능
- 향상된 처리량
  - I/O 작업 병렬 처리 가능
- 자원 공유 비용이 저렴함
- 멀티 코어에서 동시에 수행이 가능함

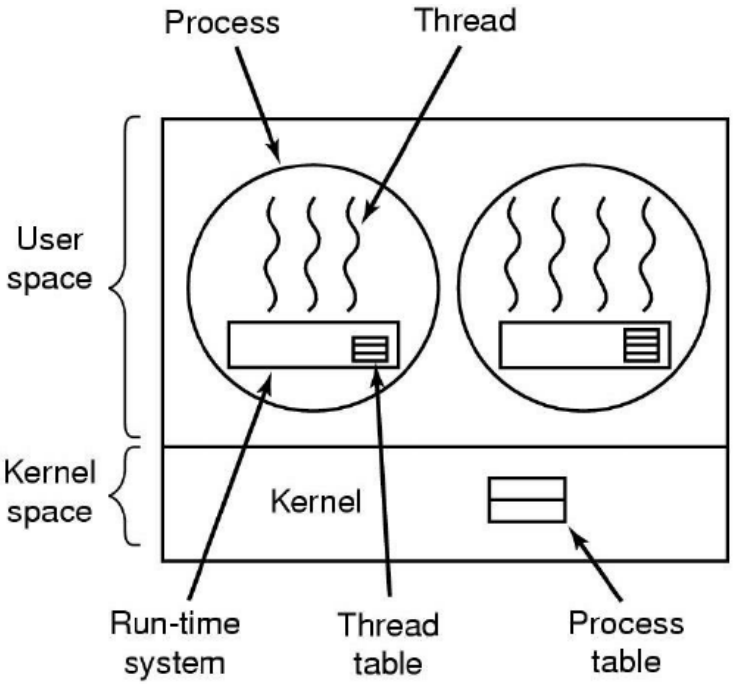
Kernel-level Threads

- 운영체제가 관리하는 쓰레드
- 모든 쓰레드의 동작이 커널에 구현되어 있음
- 쓰레드 생성 및 관리는 system call을 통해 수행 가능
- OS 스케줄러가 쓰레드 스케줄링
- Windows, Linux, Solaris, Mac OS X, etc.



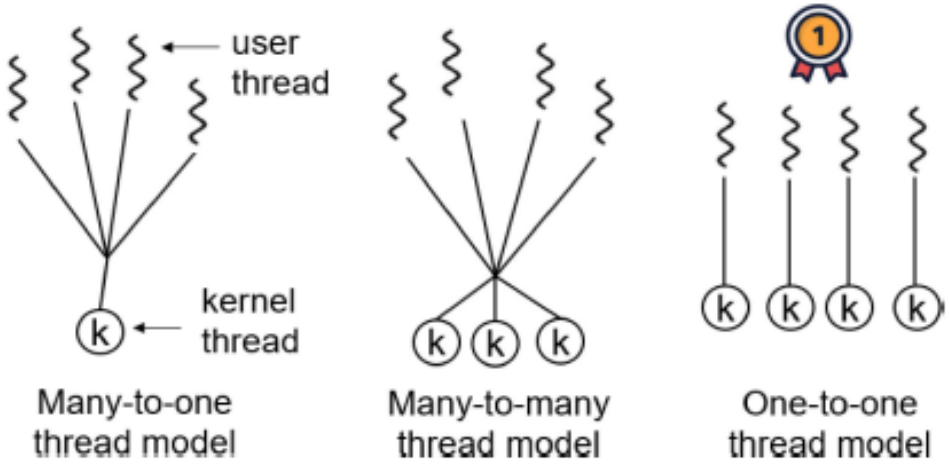
User-level Threads

- 모든 쓰레드의 동작이 사용자 레벨(user level)에 구현되어 있음
  - 라이브러리 기반 쓰레드 관리
- 운영체제에서는 쓰레드의 존재를 모름
- 모든 동작은 프로시저 콜을 통해 수행함
  - 즉, 커널의 개입이 없음
- Kernel-level Threads 보다 작고 빠름
  - 약 10-100x 빠름
- 이식성이 높음
- 어플리케이션별 최적화 가능



Thread Model

- Many-to-one model
- Many-to-many model
- One-to-one model



[Paper: Thread Evolution Kit for Optimizing Thread Operations on CE/IoT Devices]

병행성 관련 용어

- 경쟁 조건 (race condition)
  - 멀티 쓰레드가 거의 동시에 임계 영역을 실행하려고 하는 상황
- 임계 영역 (critical section)
  - 변수나 자료 구조와 같은 공유 자원을 접근하는 코드의 일부분
- 상호 배제 (mutual exclusion)
  - 하나의 쓰레드만 임계 영역에 진입할 수 있도록 보장하는 것

경쟁 조건 예제

- Example with two threads
  - counter = counter + 1 (default is 50)
  - We expect the result is 52. However,

OS	Thread1	Thread2	PC (after instruction)	%eax	counter
	before critical section		100	0	50
	mov 0x8049a1c, %eax		105	50	50
	add $0x1, %eax		108	51	50
interrupt					
save T1's state					
restore T2's state			100	0	50
		mov 0x8049a1c, %eax	105	50	50
		add $0x1, %eax	108	51	50
		mov %eax, 0x8049a1c	113	51	51
interrupt					
save T2's state					
restore T1's state			108	51	50
	mov %eax, 0x8049a1c		113	51	**51**

임계 영역 및 상호 배제 예제

- 락(Lock)
  - 임계 영역(critical section)에 진입한 쓰레드가 원자적으로 임계 영역 내부의 명령어를 수행하기 위한 방법

```
1    lock_t mutex;
2    . . .
3    lock(&mutex);
4    balance = balance + 1;
5    unlock(&mutex);
```

→ Critical section

Threads Interface

- Pthreads (POSIX Threads)
  - A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization
  - API specifies the behavior of the thread library
  - Implementation is up to development of the library
  - Common in Unix-like operating systems
    - Linux, Mac OS X, Solaris, FreeBSD, NetBSD, OpenBSD, etc.

- Microsoft Windows has its own Thread API
  - Win32/Win64 threads

Thread Creation

- How to create and control threads?

```
#include <pthread.h>

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

- `thread`: Used to interact with this thread.
- `attr`: Used to specify any attributes this thread might have.
  - Stack size, Scheduling priority, …
- `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 (Cont.)

- If `start_routine` instead required another type argument, the declaration would look like this:
  - An integer argument:

```
int
pthread_create(…, // first two args are the same
                  void*  (*start_routine)(int),
                  void*    arg);
```

  - Return an integer:

```
int
pthread_create(…, // first two args are the same
                  int  (*start_routine)(void*),
                  void*    arg);
```

Example: Creating a Thread

```
#include <pthread.h>

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);
    …
}
```

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
  - Because `pthread_join()` routine changes the value, you need to pass in a pointer to that value.

Example: Waiting for Thread Completion

```
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  }
24
```

Example: Waiting for Thread Completion (Cont.)

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

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   }
```

Example: Simpler Argument Passing to a Thread

- Just passing in a single value

```
1   void *mythread(void *arg) {
2       int m = (int) arg;
3       printf("%d\n", m);
4       return (void *) (arg + 1);
5   }
6
7   int main(int argc, char *argv[]) {
8       pthread_t p;
9       int rc, m;
10      pthread_create(&p, NULL, mythread, (void *) 100);
11      pthread_join(p, (void **) &m);
12      printf("returned %d\n", m);
13      return 0;
14  }
```

Q&A