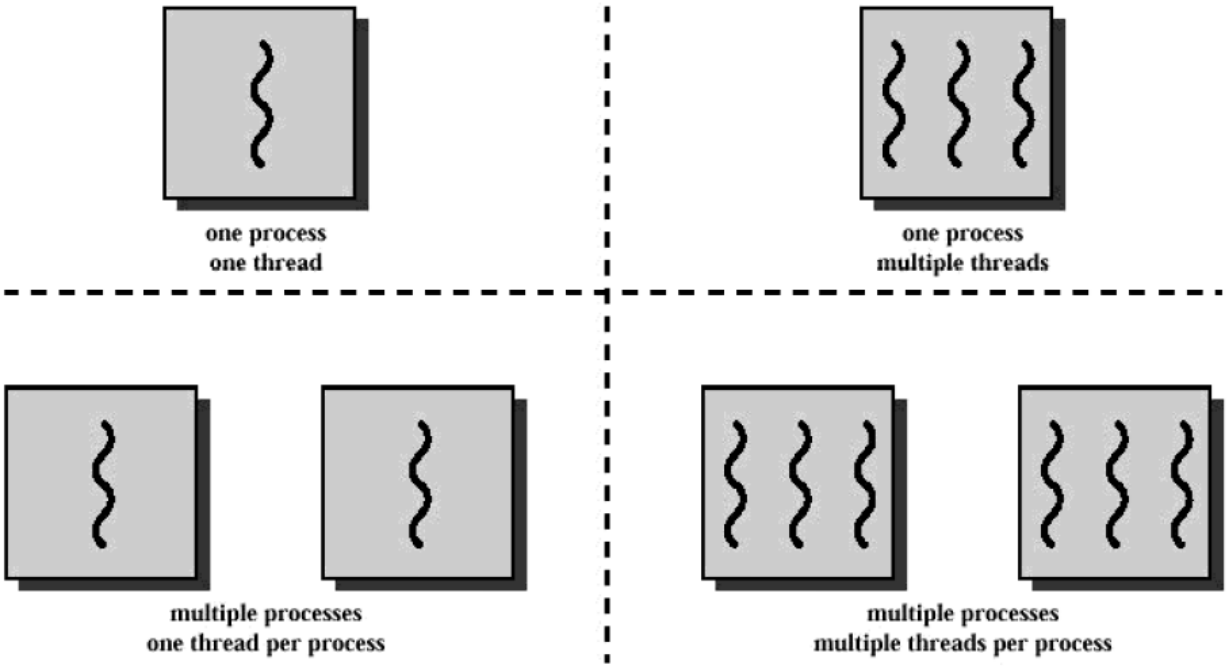
System Software Laboratory

KU KONKUK UNIVERSITY

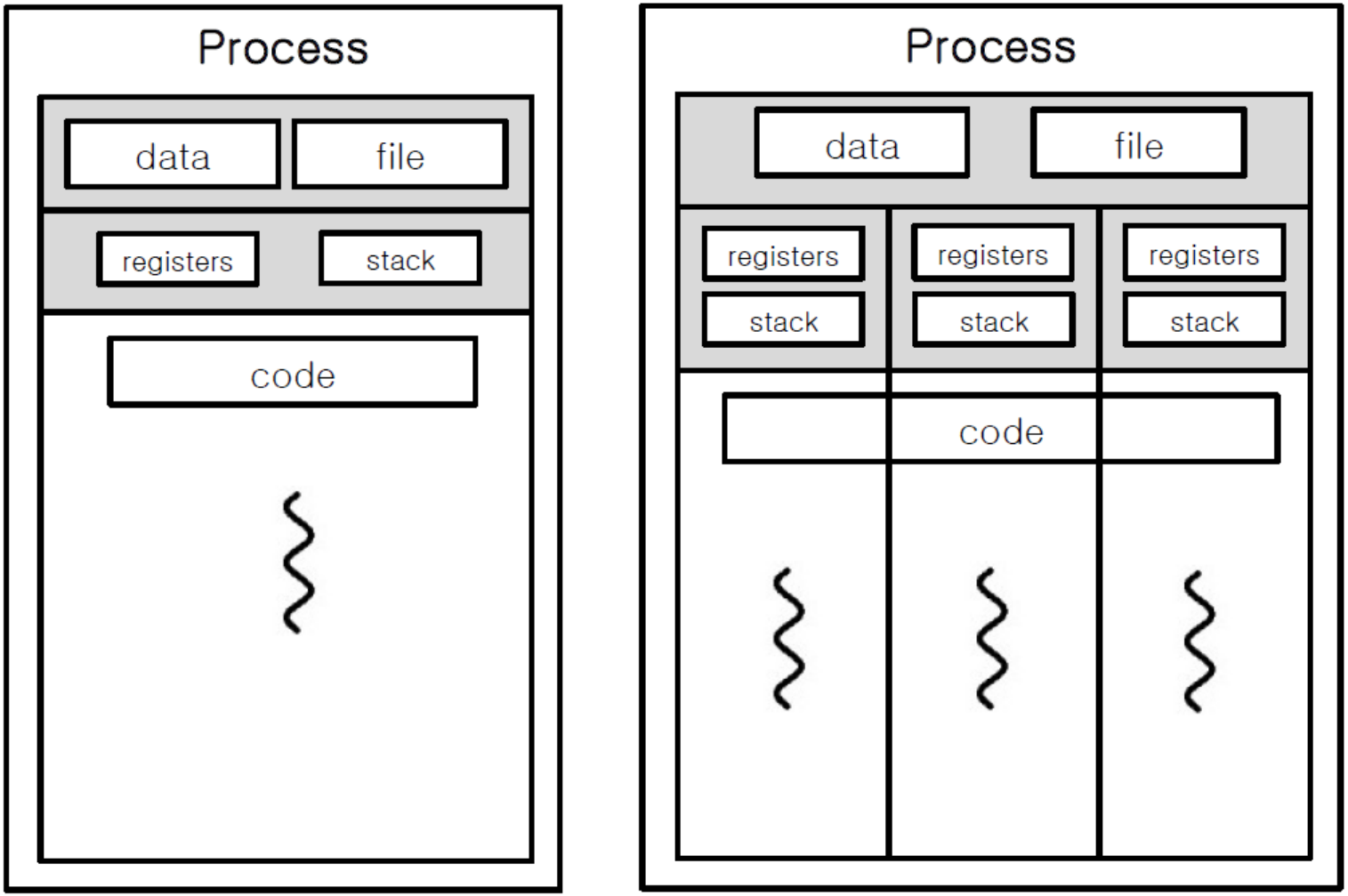
Thread Mutex

Hyun-Wook Jin
System Software Laboratory
Department of Computer Science & Engineering
Konkuk University
jinh@konkuk.ac.kr

Thread


one process
one thread
one process
multiple threads
multiple processes
one thread per process
multiple processes
multiple threads per process

Thread


Process
data
file
registers
stack
code
Process
data
file
registers
stack
registers
stack
registers
stack
code

Thread의 확인

- ps –eLF | grep 파일명
  - e 모든 프로세스
  - L thread 정보
  - f 보여질 수 있는 모든 정보

UID	PID	PPID	LWP	C	NLWP	STIME	TTY	TIME	CMD
unja	1464	1	1464	0	1	5430	396	0 21:46 ?	00:00:00 /usr/bin/VBoxClient --clipboard
unja	1466	1464	1466	0	2	5598	1868	0 21:46 ?	00:00:00 /usr/bin/VBoxClient --clipboard
unja	1466	1464	1469	0	2	5598	1868	0 21:46 ?	00:00:00 /usr/bin/VBoxClient --clipboard
unja	1489	1	1489	0	1	5430	404	0 21:46 ?	00:00:00 /usr/bin/VBoxClient --display
unja	1491	1489	1491	0	2	5595	1316	0 21:46 ?	00:00:00 /usr/bin/VBoxClient --display
unja	1491	1489	1510	0	2	5595	1316	0 21:46 ?	00:00:00 /usr/bin/VBoxClient --display
unja	1497	1	1497	0	1	5430	400	0 21:46 ?	00:00:00 /usr/bin/VBoxClient --seamless
unja	1500	1497	1500	0	2	5595	1248	0 21:46 ?	00:00:00 /usr/bin/VBoxClient --seamless
unja	1500	1497	1503	0	2	5595	1248	0 21:46 ?	00:00:00 /usr/bin/VBoxClient --seamless
unja	1506	1	1506	0	1	5430	400	0 21:46 ?	00:00:00 /usr/bin/VBoxClient --draganddrop
unja	1508	1506	1508	0	3	5724	1268	0 21:46 ?	00:00:00 /usr/bin/VBoxClient --draganddrop
unja	1508	1506	1513	0	3	5724	1268	0 21:46 ?	00:00:00 /usr/bin/VBoxClient --draganddrop
unja	1508	1506	1514	0	3	5724	1268	0 21:46 ?	00:00:00 /usr/bin/VBoxClient --draganddrop
unja	1860	1669	1860	0	1	789	924	0 21:48 pts/0	00:00:00 grep --color=auto /usr/bin/VBoxClient

UID : user ID / PID : process ID / PPID : parent process ID / LWP : thread ID / C : CPU usage / NLWP : # of thread / STIME : Time when process started / TTY : Terminal associated with the process / TIME : The amount of CPU time used by the process / CMD : Name of the process, including arguments

pthread_create()

- POSIX thread를 생성

```
#include <pthread.h>

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

Arguments	첫번째 인자 : 생성된 thread의 id를 저장할 변수의 포인터 두번째 인자 : thread의 특성을 설정할 때 사용하는데, 주로 NULL이 옴 세번째 인자 : thread가 생성되고 나서 실행될 함수가 옴(함수포인터) 네번째 인자 : 세번째 인자에서 호출되는 함수에 전달되고자 하는 인자 값
Returns	성공 시 0을 리턴 실패했을 경우, 0이 아닌 에러코드 값 리턴

pthread_join()

- thread 종료 대기

#include <pthread.h> int pthread_join(pthread_th, void **thread_return);	
Arguments	첫번째 인자 : thread의 ID가 온다. 이 ID가 종료할 때까지 실행을 지연 두번째 인자 : thread가 종료시 리턴하는 값을 받을 수 있다
Returns	성공시 0을 리턴 실패시 0이외의 에러 코드

pthread 실습

pthread_test1.c

```c
//pthread_create(), pthread_join(), pthread_self()
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

//쓰레드에서 사용할 함수
void *t_function(void *data)
{
	int id;
	int i = 0;

	id = *((int *)data);

	while(1)
	{
		//thread 식별자 출력,
		printf("(%lu) %d : %d\n", pthread_self(), id, i);
		i++;
		if(i==5)
		return (void *)i;
		sleep(1);
	}
}
```

* pthread_self()는 해당하는 스레드의 TID값을 리턴해주는 함수

pthread 실습

pthread_test1.c

```c
int main(void)
{
	pthread_t p_thread[2]; //thread ID 저장할 변수 2개
	int thr_id; //thread generation error check
	int status; //thread 종료시 반환하는 값 저장 변수
	int a = 1; //쓰레드 함수 인자
	int b = 2; //쓰레드 함수 인자

	//thread1 generation
	thr_id = pthread_create(&p_thread[0], NULL, t_function, (void *)&a);
	if(thr_id < 0)
	{
		perror("thread create error : ");
		exit(0);
	}

	//thread2 generation
	thr_id = pthread_create(&p_thread[1], NULL, t_function, (void *)&b);
	if(thr_id < 0)
	{
		perror("thread create error : ");
		exit(0);
	}

	//thread1이 종료될 때까지 main함수 종료를 기다린다. thread1이 종료시 반환하는 값 받는다.
	pthread_join(p_thread[0], (void **)&status);
	printf("return thread 0 %d\n", status);
	//thread2가 종료될  때까지 mian함수 종료를 기다린다. thread2가 종료시 반환하는 값 받는다.

	pthread_join(p_thread[1], (void **)&status);
	printf("return thread 1 %d\n", status);

	return 0;
}
```

```
gcc -o pthread_test1 pthread_test1.c -lpthread
```

pthread 참고

- pthread_t 로 정의된 thread id는 프로세스 내부에서 스레드들을 구분하기 위한 식별자
  - ps –eLF의 결과로 나오는 tid 값과 다름

```
pid_t gettid(void){
        return syscall( __NR_gettid );
}

void *t_function(void *data){
        int id ;
        int i = 0;

        id = *((int*)data);

        while(1){
                printf("(%lu) %d : %d \n", pthread_self(), id, i);
                printf("%d : tid \n", gettid());
                i++;
                if(i == 5) return (void*)i;
                sleep(1);
        }
}
```

pthread 실습 순서

- 코드 작성
- 컴파일
  - #gcc –o pthread_test1 pthread_test –lpthread
- 실행
- 실행되고 있는 스레드 리스트 확인
  - #ps –eLF | grep pthread_test1

Thread 동기화

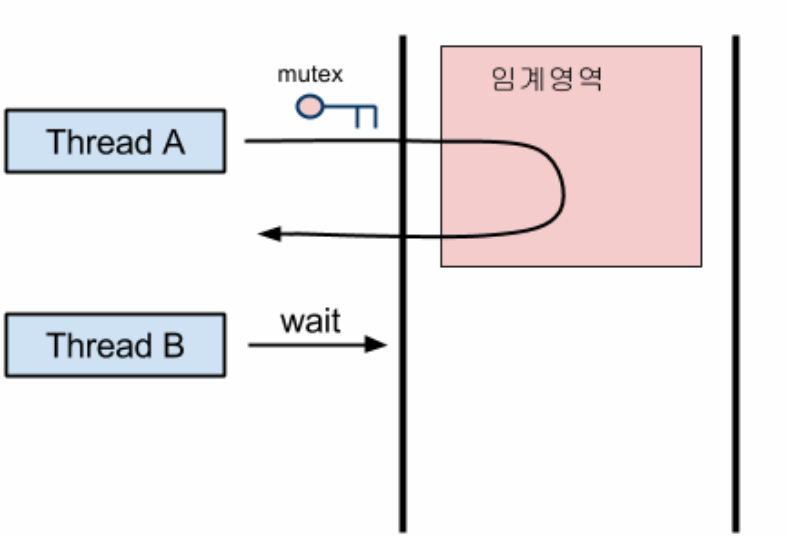
- 공유자원 영역에 접근하는 객체들의 진입시간을 제어

예시)
1. 전역 변수 count, 두 개의 쓰레드 A, B
2. count = 0
3. A가 count값 0을 읽는다
4. B가 count값 0을 읽는다
   - A가 연산을 하기 전에 B가 접근
5. A가 count + 1 연산을 하고 값을 쓴다. count = 1
6. B가 count + 1 연산을 하고 값을 쓴다. count = 1

본래 순서대로 연산을 하였으면, count 값은 2가 되어야함

Thread 동기화

- 보호 해야할 공유 자원이 있는 공간을 임계영역 (Critical section) 이라고 한다.
  - 공유 자원에 접근하는 부분
  - 임계영역에 들어가고 다른 객체로부터의 접근을 막기 위해 키(mutex)가 필요



POSIX MUTEX

- thread의 동기화

#include <pthread.h>	
MUTEX 초기화	int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *mutexattr);
Lock	int pthread_mutex_lock(pthread_mutex_t *mutex);
Trylock	int pthread_mutex_trylock(pthread_mutex_t *mutex);
unlock	int pthread_mutex_unlock(pthread_mutex_t *mutex);
MUTEX 해제	int pthread_mutex_destroy(pthread_mutex_t *mutex);

POSIX MUTEX 실습

pthread_test2.c

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

int num = 0; //동기화 할 공유 데이터
pthread_mutex_t mutex;

void *thread_func(void *arg)
{
	int i = 0;
	for(i=0; i < 5; i++)
	{
		pthread_mutex_lock(&mutex); // lock을 걸어서 다른 스레드 접근 금지
		num++; // 공유 변수 증가
		printf("%s\t : %d\n",(char*)arg, num);
		pthread_mutex_unlock(&mutex); //lock해제
		sleep(1);
	}
}
```

POSIX MUTEX 실습

pthread_test2.c

```
int main()
{
        pthread_t p_thread[2];
        char *thread1 = "Thread_A";
        char *thread2 = "Thread_B";
        void *t_return = NULL;

        pthread_mutex_init(&mutex, NULL); // mutex 초기화

        pthread_create(&p_thread[0], NULL, thread_func, thread1);
        pthread_create(&p_thread[1], NULL, thread_func, thread2);

        pthread_join(p_thread[0], &t_return);
        pthread_join(p_thread[1], &t_return);

        pthread_mutex_destroy(&mutex); //mutex 해제

return 0;
}
```

gcc –o pthread_test2 pthread_test2.c –lpthread

실습 문제 I – 토마스와 친구들

- 토마스와 친구들이 여행을 가는 중 하나의 기차만 지나갈 수 있는 다리를 발견했다
  - Tomas, James, Percy
- 모두 다리를 지나가되 서로 경쟁하여 부딪히면 서로 사이가 나빠질 수 있다. 안전한 다리로 만들어주자

```
void * is_railroad_usable(void * name)
{
        printf("%s cross a bridge₩n", name);
        sleep(1); //다리 건너는중..
        printf("%s is crossed₩n", name);
}
```

```
➜ Mutex ./tomas
Tomas cross a bridge
James cross a bridge
Percy cross a bridge
Percy is crossed
Tomas is crossed
James is crossed
```

```
➜ Mutex ./tomas
James cross a bridge
James is crossed
Tomas cross a bridge
Tomas is crossed
Percy cross a bridge
Percy is crossed
```

- 친구들이 다리를 무사히 빠져나가도록 만들어보자

실습 문제 Ⅱ- 철학자는 배고프다

- 철학자는 다음과 같은 과정을 통해서 식사함
  - 1. 왼쪽 포크가 사용 할 수 있을 때 까지 생각한다
  - 2. 오른쪽 포크가 사용 할 수 있을 때 까지 생각한다
  - 3. 왼쪽과 오른쪽 포크 모두 사용 가능하면 1초간 식사를 한다
  - 4. 식사를 마치면 오른쪽 포크를 내려놓는다
  - 5. 왼쪽 포크를 내려놓는다
  - 6. 다시 1번으로 간다

- 실습 Tip
  - 철학자를 Thread로 만든다
  - 포크를 Mutex로 만든다
  - pthread_mutex_trylock() 활용