THREADS

Jo, Heeseung

Multi-threaded program

빠른 실행

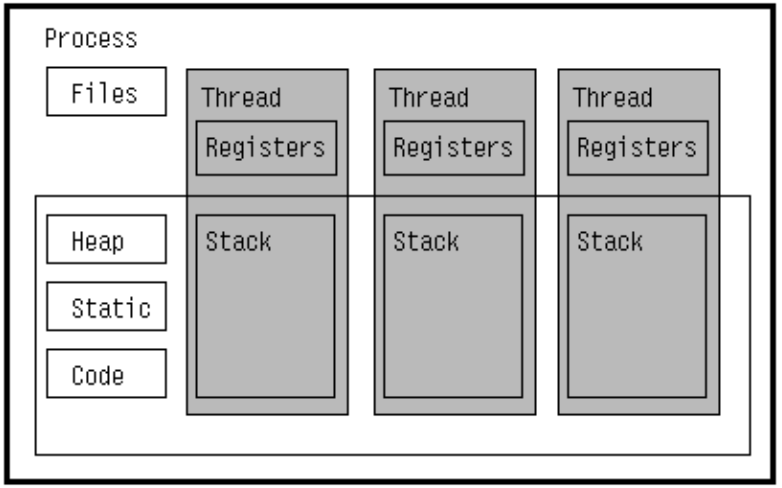
- 프로세스를 새로 생성에 드는 비용을 절약

데이터 공유

- 파일, Heap, Static, Code의 많은 부분을 공유

CPU를 보다 효율적으로 활용

- 코어가 여러 개일 경우 코어에 thread를 할당하는 방식



Multi-threaded program

Pros.

- 대량의 데이터 처리에 적합
  - CPU 자원을 효율적으로 사용
  - 멀티 프로세스 방식에 비해서 빠른 thread 생성
- 데이터 교환이 쉬움
  - IPC(inter process communication)를 사용하지 않고, 데이터를 교환할 수 있음

Cons.

- 프로그래밍 난이도 상승
  - 비직관적
  - 문맥의 흐름을 예상하기 어려움
- 디버깅이 어려움
- 제대로 만들기가 어려움
  - 병렬 프로그래밍, 공유 자원 관리는 높은 기술 숙련도를 요구

POSIX Thread : pthread

POSIX 표준을 따르는 pthread

- Multi thread 프로그래밍을 위한 API를 제공

뛰어난 호환성

- 거의 모든 운영체제를 지원

CPU 각 벤더에서도 독자적인 thread API를 제공

- pthread에 비해서 뛰어난 성능을 보여주지만,
- 이식성과 유지/보수 상의 문제로 대부분 pthread를 이용

Posix Threads (pthreads) Interface

pthreads: Standard interface for ~60 functions that manipulate threads from C programs

- Creating and reaping threads
  - pthread_create()
  - pthread_join()
- Determining your thread ID
  - pthread_self()
- Terminating threads
  - pthread_cancel()
  - pthread_exit()
  - exit() [terminates all threads] , RET [terminates current thread]
- Synchronizing access to shared variables
  - pthread_mutex_init
  - pthread_mutex_[un]lock
  - pthread_cond_init
  - pthread_cond_[timed]wait
  - pthread_cond_signal

Thread Create

```
#include <pthread.h>
int pthread_create(pthread_t *thread, pthread_attr_t *attr,
          void *(*start_routine)(void *), void *arg);
```

- thread : pthread_t structure (thread id)
- attr : pthread_attr_t structure
- start_routine : Thread 함수
- arg : Thread 함수로 넘길 매개 변수

Thread Create

```
#include <pthread.h>
int pthread_join(pthread_t thread, void **value_ptr);
```

- 해당 thread가 종료할 때까지 대기
- thread : pthread_t structure
- value_ptr : return value pointer

```
#include <pthread.h>
int pthread_detach(pthread_t thread);
```

- 해당 메인 thread로 부터 분리함
- Thread 종료와 동시에 자원 회수

The Pthreads "hello, world" Program

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

void *threadfunc(void *vargp);

/* thread routine */
void *threadfunc(void *vargp) {
  sleep(1);
  printf("Hello, world!\n");
  return NULL;
}

int main() {
  pthread_t tid;

  pthread_create(&tid, NULL, threadfunc, NULL);
  printf("main\n");
  pthread_join(tid, NULL);
  printf("main2\n");
  sleep(2);
  return 0;
}
```

```
# gcc ex.c -lpthread
# ./a.out
main
Hello, world!
main2
```

pthread 라이브러리를 linking

The Pthreads "hello, world" Program

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

void *threadfunc(void *vargp);

/* thread routine */
void *threadfunc(void *vargp) {
  sleep(1);
  printf("Hello, world!\n");
  return NULL;
}

int main() {
  pthread_t tid;

  pthread_create(&tid, NULL, threadfunc, NULL);
  printf("main\n");
  pthread_detach(tid);
  printf("main2\n");
  sleep(2);
  return 0;
}
```

```
# gcc ex.c -lpthread
# ./a.out
main
main2
Hello, world!
```

Thread Mutex

```
#include <pthread.h>
pthread_mutex_t mutex;
int pthread_mutex_init(pthread_mutex_t *restrict mutex,
          const pthread_mutexattr_t *restrict attr);
int pthread_mutex_destroy(pthread_mutex_t *mutex);

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;    // 매크로
```

```
#include <pthread.h>
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

공유 data를 사용하는 threads

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

#define MAX_THREAD 20

void *countall(void *data)
{
        int *count = (int *)data;
        int i;
        pthread_t thread_id = pthread_self();

        for (i=0; i<100; i++)       // for (i=0; i<10000; i++)
        {
                *count = *count+1;
        }
}
```

공유 data를 사용하는 threads

```
int main(int argc, char **argv)
{
        pthread_t thread_id[MAX_THREAD];
        int i = 0;
        int count = 0;

        for(i = 0; i < MAX_THREAD; i++)
        {
                pthread_create(&thread_id[i], NULL, countall, (void *)&count);
        }

        for(i = 0; i < MAX_THREAD; i++)
        {
                pthread_join(thread_id[i], NULL);
        }

        printf("Main Thread : %d\n", count);
        return 0;
}
```

```
# gcc ex.c -lpthread
# ./a.out
Main Thread : 2000
# ./a.out
Main Thread : 1957
```

pthread_mutex_lock/unlock

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

#define MAX_THREAD 20

void *t_func(void *data)
{
        int *count = (int *)data;
        int i;
        pthread_t thread_id = pthread_self();

        for (i=0; i<10000; i++)
        {
                pthread_mutex_lock(&m_lock);
                *count = *count+1;
                pthread_mutex_unlock(&m_lock);
        }
}
```

pthread_mutex_lock/unlock

```
pthread_mutex_t m_lock;

int main(int argc, char **argv)
{
        pthread_t thread_id[MAX_THREAD];
        int i = 0;
        int count = 0;

        if(pthread_mutex_init(&m_lock, NULL) != 0)
        {
                perror("Mutex Init failure");
                return 1;
        }

        for(i = 0; i < MAX_THREAD; i++)
        {
                pthread_create(&thread_id[i], NULL, t_func, (void *)&count);
        }

        for(i = 0; i < MAX_THREAD; i++)
        {
                pthread_join(thread_id[i], NULL);
        }
        pthread_mutex_destroy(&m_lock);
        printf("Main Thread : %d\n", count);
        return 0;
}
```

```
# gcc ex.c -lpthread
# ./a.out
Main Thread : 2000
```

Exercise

Ex.

- 구구단을 출력하는 threaded program 작성
- thread function이 한번 불리면 한 단을 출력
- 한번에 한 단씩 섞이지 않도록 출력 (pthread_join 사용)

Ex.

- Parameter로 thread의 수를 입력받고, 해당 수만큼 thread를 생성
- 각 thread는 생성된 순서의 번호와 자기 thread id를 출력
- 생성된 순서의 번호는 thread function의 parameter로 넘길 것
- thread 생성 순서 번호가 섞이지 않도록 할 것

```
$ ./a.out 3

thread 1 10100
thread 3 10102
thread 2 10101
```

pthread 동기화

```
#include <pthread.h>
pthread_cond_t cond;
int pthread_cond_init(pthread_cond_t *restrict cond,
          const pthread_condattr_t *restrict attr);
int pthread_cond_destroy(pthread_cond_t *cond);

pthread_cond_t cond = PTHREAD_COND_INITIALIZER;       // 매크로
```

```
#include <pthread.h>
int pthread_cond_wait(pthread_cond_t *restrict cond,
          pthread_mutex_t *restrict mutex);
int pthread_cond_signal(pthread_cond_t *cond);
```

restrict ??

- pointer가 가르키는 메모리가 서로 다르다는 의미
- 컴파일러의 최적화를 도와줌

pthread_cond_wait/signal

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

pthread_mutex_t mut;
pthread_cond_t cond;

void *func(void *data)
{
    printf("before wait\n");
    pthread_mutex_lock( &mut );
    pthread_cond_wait( &cond, &mut );
    printf("after wait\n");
    pthread_mutex_unlock( &mut );

    return 0;
}
```

pthread_cond_wait/signal

```
int main(int argc, char *argv[])
{
    int res = 0;
    res = pthread_mutex_init(&mut, NULL);
    res = pthread_cond_init(&cond, NULL);

    pthread_t pt;
    pthread_create(&pt, NULL, func, NULL);

    sleep(2);
    pthread_cond_signal(&cond);

    pthread_join(pt, NULL);

    pthread_mutex_destroy(&mut);
    pthread_cond_destroy(&cond);

    return 0;
}
```

```
# gcc ex.c -lpthread
# ./a.out
before wait
after wait
```

pthread_* functions

수 많은 pthread 지원을 위한 함수들이 존재

Manual page 참고

- man pthreads

Exercise

Ex.

- /tmp/ssuis_ref.fna라는 파일에는 2M 분량의 text string이 들어있고, 각 라인은 60개의 character로 구성되어 있다.
  - 자기 폴더로 복사하고 사용할 것
- 이중에서 "aaccggtt"가 포함된 라인을 출력하는 findst 프로그램을 작성
- findst는 파라메터로 두 개를 입력받는다.
  - 1번째 파라메터: 병렬화할 수 (ex. 4)
  - 2번째 파라메터: 파일의 이름 (ex. ssuis_ref.fna)
- 병렬화할 수 만큼 thread를 생성하여 파일내에서 찾고 결과를 출력
- gettimeofday를 이용하여 수행시간을 출력한다.

- ssuis_ref.fna는 33459 라인을 가지고 있고, 결과는 15 라인

Exercise

Ex.

- findst 4 ssuis_ref.fna

```
gtacagacacttgagccggctactcctcaagaaacttttaaaccggttcattctgatata
gtatctacatagaactttcagtgtaaaaaatcccaaaaaaccggttgacaattgccaaag
cagtttttcgccaaatttcataaaaaaatccaaaaaaccggttgacaattgccaaagtag
aaagaaccggttttagaacatattcattattatcttgatagtggggaatttgtgactttg
aaactagataaaactgaataaccggttgatattgggtagaagggaaactgactgttcaat
tcaaaaaaccggttgacaatgtatataagtcatgatagaataaatgagttgtctcttgag
taattgtaccaatgaaattctagcaagtctaggacaaatgtatattgcagataaccggtt
tttgcagctgttttatcattttcaaaaaaccggttgacaatgtatataagtcgtgataga
gaaaaatacttacggaaccggttccttcatcgtgatgaatacaggtgaagagatgcagtt
gtattgtaatatagtaaaccggttttgtcaacggtttttatcatttatttaaatacttt
taatctttgcttgaaaccggttctggtacaacaagtccttgctccttagcttgtttccaa
ttctcaacaaccggttcatcaatccctttctctaagcgagatagcataggccctgcacca
tcatccattcctgccatccaagcaagatttggttcctcaaaaccggttaccttgatggtt
cttgatgaaaccggttcatcccttgacctaccagaagaccaaagtccctttgcccttatc
cacaaccggttcttcaaacttaatttcttttggcttcttaggttgctttttagcttcctg
Execution time: 1.115533s
```