Operating system assignment 3 – My thread

201421104 허준영

1. 구현한 것

-tcb structure

```
// Thread control block
struct tcb {
  // TODO: Fill your own code
  int finished; // 0이면 끝나지 않음, 1이면 끝남.
  ucontext_t* ucp;
  int stack[16384];
};
```

종료를 확인하는 finished, 진행 중인 context를 저장할 ucp, 그리고 makecontext에 사용될 stack을 넣었습니다. Thread id의 경우 tcb를 배열로 만들면 index 자체가 id라 넣지 않았습니다.

-nextTcb()

```
static int nextTcb() {
  if (sched_policy == MYTHREAD_RR) {
    // TODO: Fill your own code
    int nextTcb;
    if(n_tcbs == 0)
      return 0;
    else{
      nextTcb = (current_thread_id+1)%(n_tcbs+1);
      while(tcbs[nextTcb]->finished == 1)
        nextTcb = (nextTcb+1)%(n_tcbs+1);
      return nextTcb;
    }
  }
  else if (sched_policy == MYTHREAD_FIFO) {
    // TODO: Fill your own code
    //?
  }
}
```

Policy에 따라 context switching이 일어난다면 다음 실행될 tcb의 id를 return 해주는 함수입니다. FIFO에서는 create가 먼저 된 thread가 먼저 실행되기 때문에 따로 건드리지 않아도 될 것 같아 비워두었습니다.
RR에서는 finished가 1인 thread들은 time quantum을 가질 필요가 없어 while문을 통해 finished가 0인 thread를 고르도록 하였습니다.

-tick()

```
//signal handler
static void tick(int sig) {
  // TODO: Implement your own code
  // printf("\ntick\n");
  if(sched_policy == MYTHREAD_RR){
    //context switchingint prev_thread_id = current_thread_id;
    int prev_thread_id = current_thread_id;
    // printf("prev id : %d\n", prev_thread_id);
    current_thread_id = nextTcb();
    // printf("current id : %d\n", current_thread_id);
    if(prev_thread_id == current_thread_id)
      return;
    else{
      // printf("swap\n");
      swapcontext(tcbs[prev_thread_id]->ucp, tcbs[current_thread_id]->ucp);
    }
  }

  // HINT: 이 함수에서 nextTcb() 함수를 통해 scheduling 하고, 해당 schedule 함수를 실행.
}
```

Signal handler로써 timer가 signal을 던지면 실행됩니다. 마찬가지로 FIFO의 경우, 선입선출이기 때문에 건드리지 않았습니다. RR의 경우, 현재 실행중인 thread의 id를 prev_thread_id에 저장하고, nextTcb()의 return값과 비교하여 같으면 그대로 함수를 실행하고, 아닐 경우 다음 tcb의 id로 context switching을 하도록 하였습니다.

-mythread_init()

```
void mythread_init(enum mythread_scheduling_policy policy)
{
  sched_policy = policy;

  // TODO: Implement your own code
  //initialize tcbs
  for(int i = 0; i < 1024 ; i++){
    tcbs[i] = (struct tcb*)malloc(sizeof(struct tcb));
    tcbs[i]->finished = 0;
    tcbs[i]->ucp = (ucontext_t*)malloc(sizeof(ucontext_t));

  }

  //timer
  // printf("set timer\n");
  memset(&ticker,0,sizeof(ticker));

  sigemptyset(&ticker.sa_mask);
  ticker.sa_handler = &tick;

  // sigaction(SIGVTALRM,&ticker,NULL);
  sigaction(SIGALRM,&ticker,NULL);

  time_quantum.it_value.tv_sec = 0;
  time_quantum.it_value.tv_usec = 1000;

  time_quantum.it_interval.tv_sec = 0;
  time_quantum.it_interval.tv_usec = 1000;
  // printf("current id : %d\n",current_thread_id);

  // setitimer(ITIMER_VIRTUAL, &time_quantum, NULL);
  setitimer(ITIMER_REAL, &time_quantum, NULL);
  // printf("finish timer setting\n");
  //

  //save main context
  getcontext(tcbs[0]->ucp);

  if(sched_policy == MYTHREAD_RR)
    printf("\nRR Scheduling\n");
  else
    printf("\nFIFO Scheduling\n");
  // HINT: 이 함수에서 tick 에 대해서 timer signal를...
}
```

Scheduling policy를 정해주고, time quantum 설정, timer setting, main context 저장 및 tcb들을 초기화하였습니다.

-new_stub()

```
void new_stub(void (*stub)(void*), void* args, struct tcb** tcbs, int tid){
  printf("do stub\n");
  stub(args);
  tcbs[tid] -> finished = 1;
  swapcontext(tcbs[tid]->ucp, tcbs[0]->ucp);
  current_thread_id = 0;
}
//
```

Tcb의 종료 여부인 finished를 함수 실행이 종료되면 1로 설정해주기 위해 구현하였습니다.

-mythread_create()

```
int mythread_create(void (*stub)(void*), void* args)
{
  // printf("\nthread creating\n");
  int tid = -1; // Thread ID
  // TODO: Implement your own code
  n_tcbs++;
  // printf("n_tcbs : %d\n", n_tcbs);
  tid = n_tcbs;
  // printf("tid : %d\n", tid);

  // printf("getcontext\n");
  getcontext(tcbs[tid]->ucp); //save new thread context
  // printf("save context\n");

  // printf("makecontext\n");
  tcbs[tid]->ucp->uc_stack.ss_sp
   = tcbs[tid]->stack;
  tcbs[tid]->ucp->uc_stack.ss_size
   = sizeof(tcbs[tid]->stack);
  tcbs[tid]->ucp->uc_link = &tcbs[0];
  makecontext(tcbs[tid]->ucp, new_stub, 4, stub, args, tcbs, tid);
  // printf("make\n");
  current_thread_id = tid;
  swapcontext(tcbs[0]->ucp, tcbs[tid]->ucp);

  return tid;
}
```

Thread를 생성해주는 함수. Thread가 종료되었다고 n_tcbs가 줄게 설계하지 않았기 때문에 thread가 추가되면 thread의 총 개수를 id로 합니다. Makecontext로 함수 정보가 tcb의 context에 들어가도록하고, swapcontext를 통해서 실행하고 thread의 id를 return하도록 구현하였습니다..

-mythread_join()

```
void mythread_join(int tid)
{
  // printf("join\n");
  int prev = current_thread_id;

  if(prev == tid){
    while(tcbs[current_thread_id]->finished == 0){
      if(tcbs[current_thread_id]->finished == 1)
        break;
    }
  }
  else{
    if(tcbs[tid]->finished == 1){
      return;
    }
    // printf("swap and join\n");
    current_thread_id = tid;
    swapcontext(tcbs[prev]->ucp, tcbs[current_thread_id]->ucp);
    while(tcbs[current_thread_id]->finished == 0){
      if(tcbs[current_thread_id]->finished == 1)
        break;
    }
  }
  // printf("finish join\n");

}
```

현재 실행중인 thread의 id를 prev에 저장하고 tid와 비교하여, 같으면 finished가 1일 때, loop를 빠져나가고, 다를 경우, 이미 thread가 끝나있다면 함수 종료, 아니라면 tid에 해당하는 thread로 context switching하여 thread의 finished가 1로 바뀔 때까지 loop를 돌도록 구현하였습니다.

2. 배운 것

Pthread를 사용하지 않고, user level threading와 context switching을 직접 구현해보니, multi-thread의 scheduling 방식인 Round Robin과 FIFO에 대해 더 잘 이해하게 되었습니다. 프로그래밍을 할 때, memory allocation하는 것을 자꾸 잊어 힘들었지만, 이번 과제를 통해 잊지 않을 것 같고, 구글링을 통해 system call을 직접 이해하고 사용해보니 영어라고 무서워했었던 예전에 비해 거리낌없이 설명을 차근차근 읽어보게 되었습니다. 또 해결하지는 못했지만, 제 프로그램이 실행이 되기는 되는데, 종종 dead lock에 걸리거나 current thread id에 대한 synchronization문제가 있는 것 같아, 해결하려고 노력했지만 실패했습니다. 이를 직접 경험해보아, 다음부터는 신경을 많이 쓰게 될 것 같습니다. 이번 과제를 통해서 프로그래밍을 하는 데에 있어서 좋은 습관들을 들여서 정말 많은 것을 배웠습니다.

3. 피드백

정말 좋은 경험을 쌓게 해준 과제인 것 같습니다. 조교님 말씀대로 context에 대한 system call과 timer에 대한 것들을 조금만 구글링하여 이해하면 쉽진 않지만 적당히 어려운 과제였던 것 같습니다. 제 프로그램은 dead lock에 걸리거나 synchronization 문제가 있는 것 같습니다. 계속 고민해 보았으나 해결하지는 못했습니다. 그래서 게속 실행시켜 timer가 signal을 보내는 타이밍이 잘 맞아야 Round Robin scheduling을 확인할 수 있습니다. 이 점을 해결 못한 것이 너무 아쉽습니다.

Tutorial

```c
void f(void* args)
{
  const int n = *((int*)args);

  for (int i = 0; i < n; ++i) {
    if(i % 100 == 0)
    fprintf(stderr, "f\n");
  }
}

void g(void* args)
{
  const int n = *((int*)args);

  for (int i = 0; i < n; ++i) {
    if(i % 100 == 0)
    fprintf(stderr, "g\n");
  }
}
```

1. 예제 code는 다음과 같습니다.

Args들은 각각 10000으로 해놨는데, i가 100으로 나누어 떨어질 때만 출력하도록 설정하였습니다.

2. mythread_init( )에서 argument 입력에 따라 scheduling 방식을 설정합니다.

MYTHREAD_RR -> Round Robin, MYTHREAD_FIFO -> FIFO

3. FIFO를 선택한다면

```
FIFO Scheduling
f
f
f
f
f
f
f
f
f
f
g
g
g
g
g
g
g
g
g
g
ajou@ajou-VirtualBox:~/mythread$
```

4.. Round Robin을 선택한다면

이러한 결과를 얻을 수 있습니다.

(Round Robin의 경우, 너무 길어서 중간에 중복되는 구간을 삭제하였습니다.)