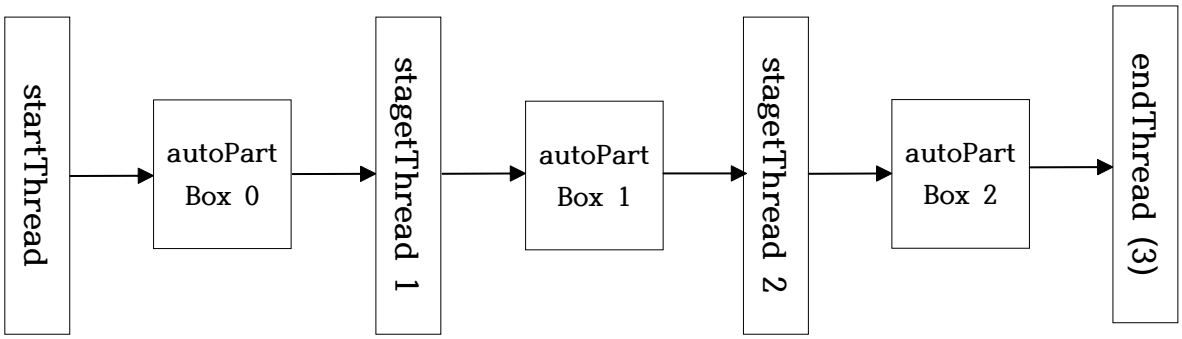
운영체제 프로그래밍 과제 2

2016.10

< Pthread를 사용한 Producer 와 Consumer 프로그래밍 응용 >

이번 프로그래밍 과제에서는 Linux하에서 Pthread를 사용 Producer/Consumer 작동을 응용하여 자동차 공장의 불량 부품 처리과정을 시뮬레이션 해보는 것입니다. 부품 처리 과정은 *N* 개의 stageThread 가 담당하고 처음과 마지막은 startThread 와 endThread 가 담당 합니다. 부품은 autoPart 구조체로 표현되고 부품의 고유번호(partNumber)를 갖습니다. 아래 그림과 같이startThread 와 endThread 사이에는 *N* 개의 stageThread가 존재하고 각각의 Thread 사이에는 autoPartBox 가 있어 autoPart를 주고받는데 사용합니다. autoPartBox는 크기가 정해져 있어 해당 크기만큼의 부품을 받으면 더 이상 받을 수가 없습니다. 각 thread의 역할은 다음과 같습니다.



1) startThread

부품(autoPart)을 만들어 내어 첫 번째 stageThread 에 전달합니다. autoPart를 할당받아 첫 번째 stageThread와 사이에 있는 autoPartBox에 보냅니다. 각 autoPart의 부품 번호는 rand()의 함수를 사용하여 절댓값으로 정합니다(abs(rand()) 함수 사용). 이 때 시드(seed) 값은 '100' 으로 설정합니다. (srand(100) 사용). 모든 부품을 다 만들어 보낸 뒤에는 ENDMARK(=-1)를 부품 번호로 갖는 autoPart를 전달합니다. 만들어야 되는 부품의 수는 인자로 받고 모든 부품 번호의 합을 pthread_exit의 status 값으로 전달합니다.

2) stageThread

N 개의 stageThread는 startThread와 endThread 사이에 존재하며 위 그림과 같이 각각의 Thread들 사이에는 autoPartBox가 있습니다. stageThread는 앞쪽 autoPartBox에서 autoPart를 받아 뒤쪽 autoPartBox 로 보냅니다. 이 과정에서 autoPart의 번호가 defectNumber 의 배수이면 해당 autoPart 는 전달하지

않습니다. defectNumber의 배수인 autoPart의 partNumber 는 모두 합하여 pthread_exit의 status 값으로 전달합니다.
ENDMARK 의 autoPart는 그대로 전달 합니다. defectNumber 와 stage ID 값 은 인자로 받습니다. stage ID 는 그림에서 보듯이 1부터 시작합니다. autoPartBox의 ID는 0부터 시작합니다.

3) endThread
마지막 autoPartBox에서 autoPart를 받습니다. 이 autoPart는 모든 stageThread에서 defectNumber의 배수 테스트를 모두 통과하고 (배수가아님) 도달한 것입니다. 인자로는 Stage 번호를 마지막 stageThread (= N+1) 라 가정하고 전달 받습니다. ENDMARK autoPart를 받으면 그동안 전달 받은 autoPart의 partNumber를 모두 합하여 pthread_exit의 status 값으로 전달합니다.

3) main()
명령어 인자로 NSTAGES BOXSIZE NPART defect_numbers 를 받습니다. 모든 인자의 값은 제한이 없습니다.
NSTAGES - stageThread의 수.
BOXSIZE - autoPartBox의 크기. 따라서 **autoPartBox에서는 autoPart를 linked list의 fifo queue로 구현 합니다.**
NPART - 생성되는 autoPart의 수
defect_numbers - NSTAGES 개수 만큼 1 번 stageThread부터 시작하여 차례로 사용하게 될 defectNumber 의 리스트

4) 주의 사항
threadpipehw.c를 사용하여 프로그램을 완성합니다.

0) Thread 간의 동기화(synchronization)을 위하여 busy waiting 방식을 사용하면 안됩니다. pthread_cond_wait(), pthread_cond_signal()을 사용합니다.
1) threadpipehw.c 에 정의 또는 선언된 변수, 구조체, 함수의 선언 등은 변경 없이 그대로 사용합니다. 추가로 변수, 구조체, 함수 등을 선언하여 사용하는 것은 자유입니다.
2) 함수 내에 있는 ***모든*** printf 문은 모두 반드시 사용합니다. 단 printf 문에 사용되는 인자의 이름은 바꾸어 사용하여도 됩니다. 함수에 사용된 인자의 이름 또한 바꾸어 사용하여도 됩니다.
3) 특히 main 함수의 assert() 문은 바꾸면 안 됩니다.
4) 주석에 이름과 학번을 반드시 기입합니다.
5) printf("*** Part Sum Information ***\n"); 와 이 이후의 출력문은

thread에서의 출력문이 다 끝난 다음에 출력되도록 barrier sync를 사용합니다. 또한 printf("*** Part Sum Information ***\n"); 이후의 문장은 main() 함수의 마지막 문장들입니다.
6) 참고로 autoPartBox가 Producer & Consumer 프로그램에서 iterm을 담는 buffer의 역할을 합니다.

<threadpipehw.c>

```
//
// Thread programming homework
// A simple thread pipeline (multiple single producer and consumer version)
// Student Name :
// Student Number :
//
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <assert.h>

struct autoPart {
        int partNumber;
        struct autoPart *next;
};

struct autoPartBox {
        int bid;  // autoPartBox id
        int SIZE;// SIZE Of autoPartBox
        int count;  // the number of autoParts in the Box
        struct autoPart *lastPart;  // Pointer to the last auto part
        struct autoPart *firstPart; // Pointer to the first auto part
        pthread_mutex_t mutex;
        pthread_cond_t full;
        pthread_cond_t empty;
};

struct stageArg {
        int sid;
        int defectNumber;
};

#define ENDMARK -1
struct autoPartBox *AutoBox;
pthread_barrier_t barrier;

void sendAutoPart(int id, struct autoPart *ap, struct autoPartBox *apBox) {
```

```
		printf("SEND:: Stage %d thread waiting on autPartBox %d full \n",id,apBox->bid);
		printf("SEND:: Stage %d sending autoPart %d to autoPartBox
%d\n",id,apBox->lastPart->partNumber,apBox->bid);
		printf("SEND:: Stage %d signals autoBoxPart %d NOT empty\n",id,apBox->bid);
}

struct autoPart *receiveAutoPart(int id, struct autoPartBox *apBox) {
		printf("RECEIVE:: Stage %d waiting on autoPartBox %d empty\n",id,apBox->bid);
		printf("RECEIVE:: Stage %d receiving autoPart %d from autoPartBox
%d\n",id,autoPtr->partNumber,apBox->bid);
		printf("RECEIVE:: Stage %d signals autoPartBox %d NOT full\n",id,apBox->bid);
		return(autoPtr);
}

// Generate autoParts and put the autoParts into the first autoPartBox
void startThread(void *ag) {

		printf("Start Thread Stage %d sending autoPart %d to autoPartBox
%d\n",0,autoPtr->partNumber,0);
		printf("Start Thread Stage %d sending ENDMARK to autoPartBox %d\n",0,0);

}

// Get autoParts from the last autoPartBox and add all of them
void endThread(void *id) {

		printf("End Thread Stage %d receiving autoPart %d from autoPartBox
%d\n",tid,autoPtr->partNumber,tid-1);
		printf("End Thread Stage %d receiving ENDMARK from autoPartBox %d\n",id,tid-1);

}

// Check autoParts from the input box and remove faulty parts
// Add all faulty parts number; Put valid autoParts into the output box
// The faulty part number is a multiple of the stage defect number
void stageThread(void *ptr) {

		printf("Stage %d receiving autoPart %d from autoPartBox
%d\n",stArg->sid,autoPtr->partNumber,stArg->sid-1);
		printf("Stage %d deleting autoPart %d\n",stArg->sid,autoPtr->partNumber);
		printf("Stage %d sending autoPart %d to autoPartBox
%d\n",stArg->sid,autoPtr->partNumber,stArg->sid);
		printf("Stage %d receiving ENDMARK from autoPartBox
%d\n",stArg->sid,stArg->sid-1);
		printf("Stage %d sending ENDMARK to autoPartBox %d\n",stArg->sid,stArg->sid);
```

```
}

int main(int argc, char *argv[]) {

        long int startThreadSum, endThreadSum, stageThreadSum;
        int i; void *status;

        startThreadSum = endThreadSum = stageThreadSum = 0;
        srand(100);


        printf("*** Part Sum Information ***\n");
        pthread_join(startTid,&status); printf("startThread sum %ld\n", status);
        startThreadSum = status;
        pthread_join(endTid,&status); printf("endThread sum %ld\n", status); e
        ndThreadSum = status;
        for(i=0; i < nStages; i++) {
                pthread_join(stageTid[i],&status); stageThreadSum += status;
                printf("Stage %d sum %ld\n", i,status);
        }

        assert(startThreadSum == (endThreadSum+stageThreadSum));

        pthread_exit(0);

}
```

***) 수행 예**

```
>tp 2 1 10 4 5
RECEIVE:: Stage 1 waiting on autoPartBox 0 empty
RECEIVE:: Stage 3 waiting on autoPartBox 2 empty
Start Thread Stage 0 sending autoPart 677741240 to autoPartBox 0
SEND:: Stage 0 sending autoPart 677741240 to autoPartBox 0
SEND:: Stage 0 signals autoBoxPart 0 NOT empty
RECEIVE:: Stage 2 waiting on autoPartBox 1 empty
Start Thread Stage 0 sending autoPart 611911301 to autoPartBox 0
RECEIVE:: Stage 1 receiving autoPart 677741240 from autoPartBox 0
RECEIVE:: Stage 1 signals autoPartBox 0 NOT full
Stage 1 receiving autoPart 677741240 from autoPartBox 0
Stage 1 deleting autoPart 677741240

/* 중간 출력 생략 */

Stage 2 receiving ENDMARK from autoPartBox 1
Stage 2 sending ENDMARK to autoPartBox 2
RECEIVE:: Stage 3 receiving autoPart 922406371 from autoPartBox 2
RECEIVE:: Stage 3 signals autoPartBox 2 NOT full
End Thread Stage 3 receiving autoPart 922406371 from autoPartBox 2
RECEIVE:: Stage 3 waiting on autoPartBox 2 empty
SEND:: Stage 2 sending autoPart -1 to autoPartBox 2
SEND:: Stage 2 signals autoBoxPart 2 NOT empty
RECEIVE:: Stage 3 receiving autoPart -1 from autoPartBox 2
RECEIVE:: Stage 3 signals autoPartBox 2 NOT full
End Thread Stage 3 receiving ENDMARK from autoPartBox 2
*** Part Sum Information ***
startThread sum 9105319910
endThread sum 3273191755
Stage 0 sum 2524404980
Stage 1 sum 3307723175
```