Department of Electrical and Computer Engineering

Operating System Assignment
COMP2006

HyoJin Cha 18611458

I confirm that this assignment is my own work

1. source code

Pthread.c

```c
//////////////////////////header////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include <semaphore.h>
////////////////////////////////////////////////////////////////////
pthread_mutex_t mutex;       // mutex
pthread_cond_t wrt;      // pthread condtion

//////////////////////global variable////////////////////////////////
int db_var= 5; //data buffer size
int readcount;//this is using as flag in this program
int db[5]; // data bufer for testing
FILE *file,*file1;
////////////////////////////////////////////////////////////////////
int initialize();
void * Writer(void *arg);
void * Reader(void *arg);
////////////////////////////메인//////////////////////////////////////////
int main(int argc, char *argv[])
{
    int rdrn,wrtn,t1,t2;//number of reder and writer and sleep time 1 and 2
    int i,j;
    rdrn = atoi(argv[1]);
    wrtn = atoi(argv[2]);
    t1 = atoi(argv[3]);
    t2 = atoi(argv[4]);
    pthread_t writer_thread[wrtn],reader_thread[rdrn];//writer와reader의 thread 선언

    initialize();//recall the initialize the mutex,condition and readcount
    file1 = fopen("data","r"); // open for writer
    for(i=0; i<wrtn; i++)
    {
        pthread_create(&writer_thread[i], NULL, &Writer, NULL);//writer thread create
    }
    for(j=0; j<rdrn; j++) //
    {
        pthread_create(&reader_thread[j], NULL, &Reader, NULL);//reader thread create
    }
    for(i=0; i<wrtn; i++)
    {
    pthread_join(writer_thread[i],NULL);//writer thread join
    }
    for(j=0; j<rdrn; j++) //
    {
    pthread_join(reader_thread[j],NULL);//reader thread join
    }
        pthread_cond_destroy(&wrt);// delete initialize wrt condition
        pthread_mutex_destroy(&mutex);// delete initialize mutex
    fclose(file1); //close for writer
    return 0;
}
int initialize()
```

```c
int initialize()
{
 pthread_mutex_init(&mutex,NULL);//mutex initialize
 pthread_cond_init(&wrt,NULL);//condition initialize
 readcount=0;//count initialize
}
void * Writer(void * arg)
{
    int pid,tid,tc=0,wc=0; // tc is temp count wc is writer counter
    tid=pthread_self();//thread id
    pid=getpid();// process id
    printf("**[writer] pthread created pid : %d tid: %d\n",pid,tid);

    pthread_mutex_lock(&mutex);
    while(!feof(file1))//loop until file is empty
    {
    while(readcount=0)//condition
    {
    pthread_cond_wait(&wrt,&mutex); //writer wait for signal and release the mutex lock
    }

    readcount=1;
    fscanf(file1,"%d\n",&db[tc]);//read the file1 and write into data buffer
    tc++;
    wc++;
    if(tc == db_var)//if temp count is same as buffer then reset the value.
    {
    tc=0;
    pthread_cond_signal(&wrt); // release the signal for wait condition
    if(!feof(file1)) // exit condtion for last writer thread becasue if there has no if condition then condition
    pthread_cond_wait(&wrt,&mutex);
    readcount=0;
    }
    sleep(1);
    }
    pthread_mutex_unlock(&mutex);//release the mutex
    printf("writer_pid : %d tid : %d has finished writing %d pieces of data to the data_buffer\n\n",pid,tid,wc);
    pthread_cond_signal(&wrt);//release the last writer condition wait

    return NULL;
}

void * Reader(void * arg)
{
    int pid,tid,tc=0,rc=0; // tc is temp count and rc is reader count
    tid=pthread_self();//thread id
    pid = getpid();//process id
    printf("**[reader] pthread created pid : %d tid : %d\n",pid,tid);

    pthread_mutex_lock(&mutex);//acquire the mutex
    readcount=1;
    file = fopen("data","r"); // open file for reader
    while(!feof(file)) // loop until file is empty
    {
```

```c
     fscanf(file,"%d\n",&db[tc]);//read the file and save into the buffer
     tc++;
     if(tc == db_var) //if temp counter is same as buffer then add to reader counter and reset
     {
      rc+=tc;
      tc=0;
      sleep(1);
     }
    }
    fclose(file); // close file for reader

    pthread_cond_signal(&wrt);//release the signal for wait condition
    pthread_mutex_unlock(&mutex);//release the mutex
    printf("reader_pid : %d tid : %d has finishied reading %dpicese of data from the data_buffer\n\n",pid,tid,rc);

    return NULL;
}
```

Process.c

```c
//////////////////////////헤더파일/////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <semaphore.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <string.h>
#include <signal.h>
//////////////////////전역변수 선언/////////////////////////////////
#define db_size 5 //data buffer size
int readcount;//this is using as flag in this program
//int *db=NULL; // data bufer for testing
int t1,t2,shmid=5;
sem_t wrt,mutex;       // wrt와mutex의 semaphore선언
FILE *file1;
////////////////////////////////////////////////////////////////////
void creat_process(int rnum,int wnum);
int initialize();

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

    int rdrn,wrtn;//number of reder and writer
    int i,j;
    rdrn = atoi(argv[1]);
    wrtn = atoi(argv[2]);
    t1 = atoi(argv[3]);
    t2 = atoi(argv[4]);
    initialize();
    file1=fopen("data","r");
    creat_process(rdrn,wrtn);

        sem_destroy(&wrt);// 초기화된것을 write삭제
        sem_destroy(&mutex);// 초기화된것을 mutex삭제
    fclose(file1);
    return 0;
}

int initialize()
{
 sem_init(&wrt,1,1);//wrt 초기화
 sem_init(&mutex,1,1);//mutex 초기화
 return 0;
}
void creat_process(int rnum, int wnum)
{
    pid_t r, w, *rchildPids=NULL, *wchildPids=NULL;
    int i,j,rwaiting,wwaiting; // tc is temp count and rc is reader count
    FILE *file;
    /* Allocate array of child PIDs: error handling omitted for brevity */
    rchildPids = malloc(rnum * sizeof(pid_t));
    wchildPids = malloc(wnum * sizeof(pid_t));
    /* Start up children */
```

```
55
56      for(i=0; i<rnum; i++)
57      {
58      if((r = fork()) == 0)
59      {
60
61      void* db;
62      int rc=0,pid;
63      file=fopen("data","r");
64
65      pid=getpid();
66      printf("**[reader] process created pid : %d\n",pid);
67
68          if((shmid=shmget((key_t)2669, 1024, IPC_CREAT|0666)) == -1)
69      {
70          perror("shmid failed");
71          exit(1);
72      }
73          if((db=shmat(shmid, (void *)0, 0)) == (void *)-1)
74      {
75      perror("shmat failed");
76          exit(1);
77          }
78
79      sem_wait(&mutex);
80
81      readcount++;
82      if(readcount==1)
83       sem_wait(&wrt);
84
85      sem_post(&mutex);
86      sleep(1);
87      while(!feof(file)) // loop until file is empty
88      {
89       fscanf(file,"%s\n",(char*)db);//read the file and save into the buffer
90       rc++;
91      }sleep(t1);
92
93      sem_wait(&mutex);
94      readcount--;
95      if(readcount==0)
96       sem_post(&wrt);
97
98      sem_post(&mutex);
99      sleep(t1);
100     printf("reader_pid : %d has finishied reading %dpicese of data from the data_buffer\n\n",pid,rc);
101     if(shmdt(db) == -1)
102     {
103         perror("shmdt failed");
104         exit(1);
105         }
106
107     fclose(file); // close file for reader
108
```

```
109
110             exit(0);
111         }
112         else
113         {
114             rchildPids[i] = r;
115         }
116         }
117  //-------------------------------------------------------------------------------------------//
118  for(j=0; j<wnum; j++)
119         {
120         if((w = fork()) == 0)
121         {
122
123         void *db;
124         int wc=0,pid,readcount=0;
125         pid=getpid();// process id
126         printf("**[writer] process created pid : %d\n",pid);
127
128
129         if((shmid=shmget((key_t)2669, 1024, IPC_CREAT|0666)) == -1)
130         {
131             perror("shmid failed");
132             exit(1);
133         }
134             if((db=shmat(shmid, (void *)0, 0)) == (void *)-1)
135         {
136         perror("shmat failed");
137             exit(1);
138         }
139
140         while(!feof(file1))//loop until file is empty
141         {
142         sem_wait(&wrt);
143
144         fscanf(file1,"%s\n",(char*)db);//read the file1 and write into data buffer
145         sleep(t2);
146         wc++;
147
148         sem_post(&wrt);
149
150         }
151
152         printf("writer_pid : %d has finished writing %d pieces of data to the data_buffer\n\n",pid,wc);
153
154         if(shmdt(db) == -1)
155         {
156             perror("shmdt failed");
157             exit(1);
158             }
159         exit(1);
160         }
161         else
162         {
```

```
163             wchildPids[i] = w;
164         }
165
166     //----------------- Wait for reader children to exit---------
167         do
168          {
169          rwaiting = 0;
170                for (i = 0; i < rnum; i++)
171           {
172                 if (rchildPids[i] > 0)
173            {
174                  if (waitpid(rchildPids[i], NULL, WNOHANG) != 0)
175            {
176                   rchildPids[i] = 0;
177                  }
178                  else
179            {
180                   rwaiting = 1;
181            }
182                 }
183                 sleep(0);
184                }
185          } while (rwaiting);
186
187         free(rchildPids);
188     //----------------- Wait for writer children to exit---------
189         do
190          {
191          wwaiting = 0;
192                for (i = 0; i < wnum; i++)
193           {
194                 if (wchildPids[i] > 0)
195            {
196                  if (waitpid(wchildPids[i], NULL, WNOHANG) != 0)
197            {
198                   wchildPids[i] = 0;
199                  }
200                  else
201            {
202                   wwaiting = 1;
203            }
204                 }
205                 sleep(0);
206                }
207          } while (wwaiting);
208
209         free(wchildPids);
210     }
211     }
212
```

Makefile

```
pthread: pthread.c
    gcc -lpthread pthread.c -o pthread

process: process.c
    gcc -lpthread process.c -o process

clean:
    rm -f pthread process
```

2.Read me

The 2 tasks (pthread and process) are coded in each xxx.c file.
So that it can be compiled by self. The makefile is attached as well so ‘make pthread’ or ‘make process’ then it is compiled itself.
When running the program just enter the program name then need to enter 4 variable which are number of reader, number of writer, sleep time for reader function and sleep time for writer function. The example is same as below.

./Pthread 5 4 1 1

3. Discussion about mutex

Mutual exclusion in block the other approaches if that is critical section.
if only using the mutex function then can control by 2 semaphore variable ,however, since using the cond_wait and cond_signal, it is really difficult to control the critical section.

```
pthread_mutex_lock(&mutex);//acquire the mutex
readcount=1;
file = fopen("data","r"); // open file for reader
while(!feof(file)) // loop until file is empty
{
 fscanf(file,"%d\n",&db[tc]);//read the file and save into the buf
 tc++;
 if(tc == db_var) //if temp counter is same as buffer then add to
 {
  rc+=tc;
  tc=0;
  sleep(1);
 }
}
fclose(file); // close file for reader

pthread_cond_signal(&wrt);//release the signal for wait condition
pthread_mutex_unlock(&mutex);//release the mutex
```

```
pthread_mutex_lock(&mutex);
while(!feof(file1))//loop until file is empty
{
while(readcount=0)//condition
{
pthread_cond_wait(&wrt,&mutex); //writer wait for signal
}

readcount=1;
fscanf(file1,"%d\n",&db[tc]);//read the file1 and write i
tc++;
wc++;
if(tc == db_var)//if temp count is same as buffer then re
{
tc=0;
pthread_cond_signal(&wrt); // release the signal for wait
 if(!feof(file1)) // exit condtion for last writer thread
on then condition wait make function keep waiting.
pthread_cond_wait(&wrt,&mutex);
readcount=0;
}
sleep(1);
}
pthread_mutex_unlock(&mutex);//release the mutex
```

Above photo is mutex part from the pthread.c, it shows different to lecture note because using the cond_wait and cond_signal, the key point is readcount(condition variable) that makes writer threads keep waiting even signaled by other threads like reader. Since reader threads finish reading then writer threads can start accessing critical section in sequence. However, cond_wait release the other writer but must one writer in condition wait so that when last before writer thread signal for release the waiting last writer thread.

```
sem_wait(&mutex);

readcount++;
if(readcount==1)
 sem_wait(&wrt);

sem_post(&mutex);
sleep(1);
while(!feof(file)) // loop unti
{
 fscanf(file,"%s\n",(char*)db);
 rc++;
}sleep(t1);

sem_wait(&mutex);
readcount--;
if(readcount==0)
 sem_post(&wrt);

sem post(&mutex);
```

```
{
sem_wait(&wrt);

fscanf(file1,"%s\n",(char*)db);/
sleep(t2);
wc++;

sem_post(&wrt);
```

Above code is mutex part from the process.c, it show exactly same as lecture note mutex for reader writer problem. Actually it is working well but one bad thing is that nobody know which process acquire the mutex lock. So that algorithm is working perfectly on the 1 by 1 reader writer problem ,however, if there has more than 3 process try accessing the critical section then this algorithm is needed additional method for control the order of mutex.

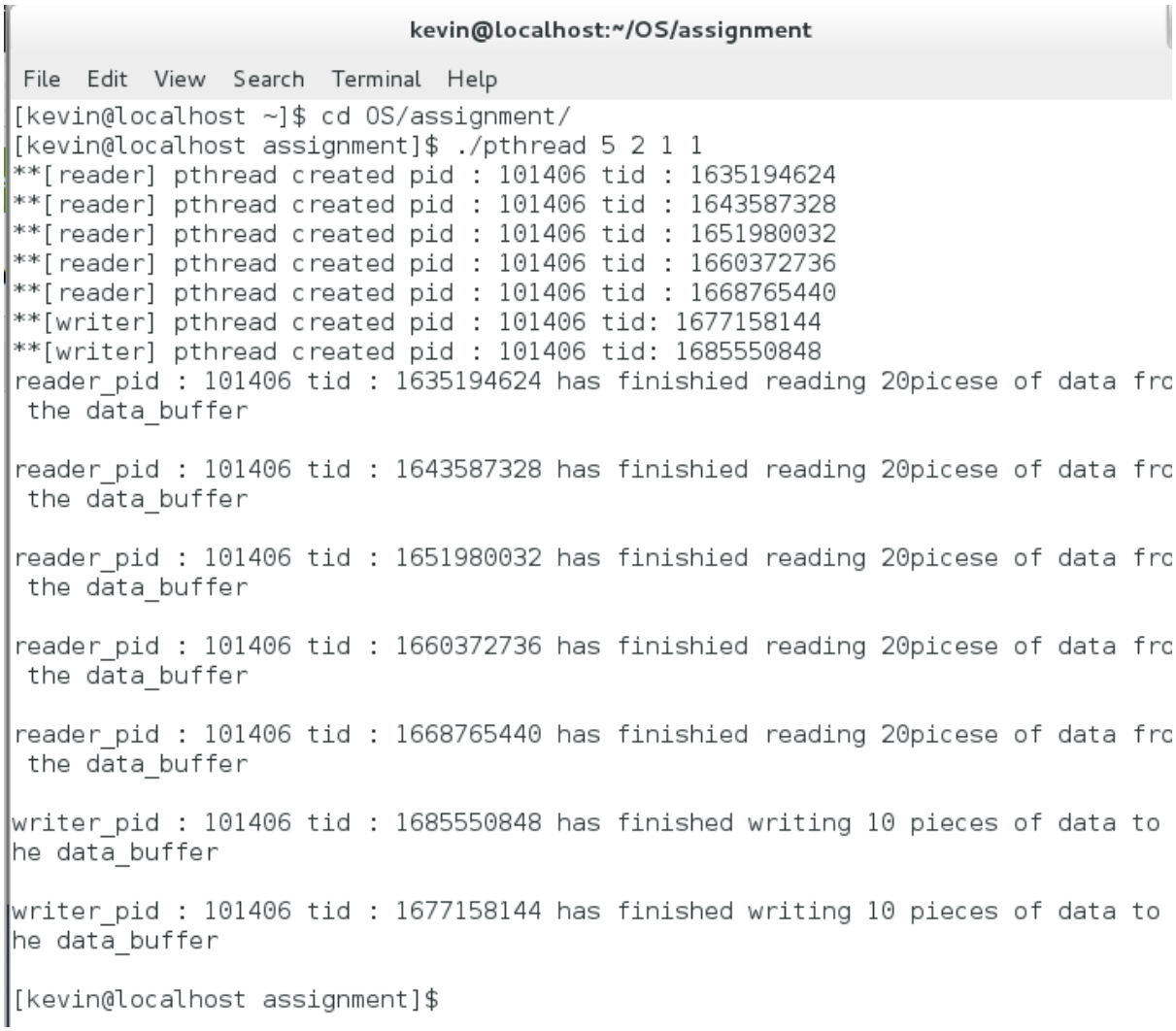
4.Descriptio of program.

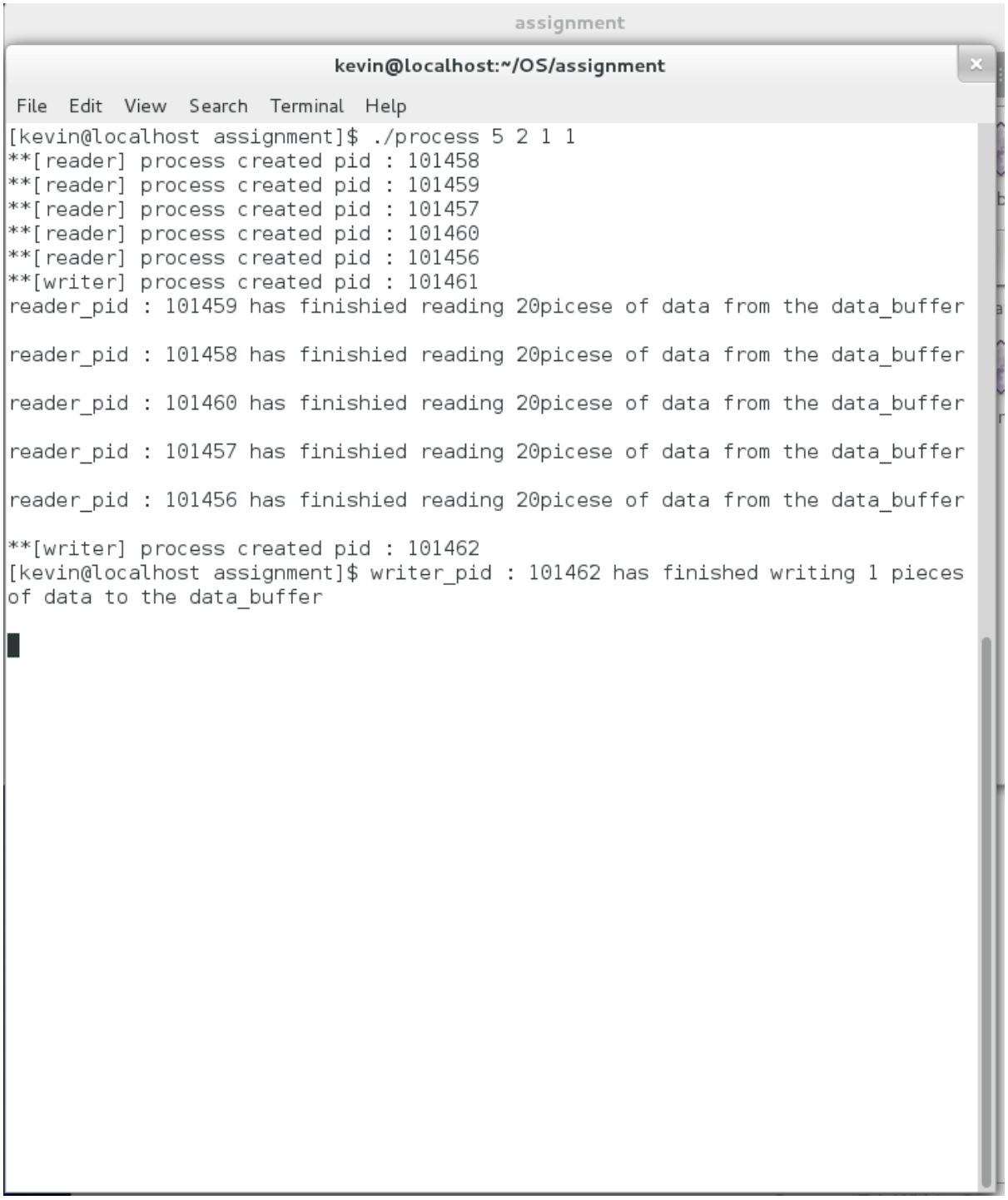
In pthread case, it runs concurrently and only one thread can access the critical section so that working properly, however, reader is not reading the critical section at same time so that is little different to the assignment's object which notice that multiple readers can read the buffer at the same time.

In process case, it has 2 unsure issue, first is shared memory part and another one is fork() function. Shared memory is created properly and attached well but it could not be integer array which contain the data_buffer form. It initialize as char* array which is string.so that could not set the buffer size so once process get in the critical section then read only one integer at time so this makes writer process cannot running properly.

Another issue is fork(), if using the for loop to make few process then those process are not running concurrently it means last child process only access the critical section without any interrupt so mutex is not needed in this case. So using the waitpid() function to make sure kill the child process when they finish their task and it makes sure do not make any zombie process but conmandline is interrupted by those child process. On the ouput of process photo on next page, some words are mixed up with command line.

5. input and output




assignment
kevin@localhost:~/OS/assignment
File Edit View Search Terminal Help
[kevin@localhost assignment]$ ./process 5 2 1 1
**[reader] process created pid : 101458
**[reader] process created pid : 101459
**[reader] process created pid : 101457
**[reader] process created pid : 101460
**[reader] process created pid : 101456
**[writer] process created pid : 101461
reader_pid : 101459 has finishied reading 20picese of data from the data_buffer
reader_pid : 101458 has finishied reading 20picese of data from the data_buffer
reader_pid : 101460 has finishied reading 20picese of data from the data_buffer
reader_pid : 101457 has finishied reading 20picese of data from the data_buffer
reader_pid : 101456 has finishied reading 20picese of data from the data_buffer
**[writer] process created pid : 101462
[kevin@localhost assignment]$ writer_pid : 101462 has finished writing 1 pieces
of data to the data_buffer