

< 2018 학년도 2 학기 - 운영체제(Operating System) 과제 #1 >

이름 : 유기성

학번 : 20120139

제출일 : 2018 년 10 월 18 일 (목)

1. Definition of the Problem

이번 과제는 Mutual Exclusion 의 대표적인 예시인 Producer – Consumer Problem 을 Semaphore 를 활용해서 구현하는 것이다.

이 과제는 크게 세 가지 부분으로 나뉘어진다.

1. fork() - wait() 함수를 호출하여 Multi-Processing 으로 producer – consumer C 코드를 구현한다.
2. 프로세스 간 통신(IPC)를 위해서 POSIX Shared Memory 를 사용한다.
3. 프로세스를 서로 동기화하는 데 POSIX Semaphore 를 활용한다.

2. Architecture of the Solution

구현 과정을 요약하면 다음과 같다.

1. Producer 기능을 하는 함수를 producer_semaphore.c 에 작성한다.
2. Consumer 기능을 하는 함수를 consumer_semaphore.c 에 작성한다.
3. 두 프로세스 사이의 공유영역에 저장하기 위한 구조체를 buffer.h 에 작성한다.
4. 이 구조체 안에 세마포어 변수도 포함시킨다.
5. main 함수를 구현해서 producer_consumer_shared_memory_main.c 에 작성한다.
 - 5-1. main 함수 내에서 shm_obj 를 생성하고 초기화한다.
 - 5-2. main 함수에서 fork()를 호출한다.
 - 5-3. parent process 는 producer 로 사용하고 child process 는 consumer 로 사용한다.
 - 5-4. child process 가 종료 하기 전에 자신의 메모리 영역을 정리한다.
 - 5-5. parent process 에서 wait()를 호출해서 child process 가 종료할 때까지 대기한다.
 - 5-6. parent process 의 메모리 영역을 정리한다.
 - 5-7. shm_obj 를 소멸한다.
6. 빌드를 위한 CMakeLists.txt 파일을 작성한다.
 - 6-1. librt.so 모듈을 사용하기 위해서 컴파일 옵션으로 “-lrt” 를 추가한다.

3. Encountered Obstacles and Realizations

(1) 이번 과제를 하면서 프로세스와 스레드의 차이를 명확하게 이해할 수 있었다.

하나의 프로세스에서 분기한 여러 개의 스레드는 메모리 영역 중에서 Text, Data, Stack 을 공유한다.

그러나 여러개로 분기한 프로세스는 오직 Text 만 공유한다.

이걸 잠시 잊고 있어서 처음에는 코드를 제대로 작성하지 못할 뻔 했지만 이를 깨닫고 난 뒤로 올바른 방향을 나아갈 수 있었다.

(2) CMakeLists.txt 를 작성하는 과정에서 많은 시간이 소요되었다. 평소에 접해보지 않았던 영역이었기 때문이다. 이번 과제를 위해서 작성한 코드를 제대로 컴파일하기 위해서는 [-lrt] 옵션을 커맨드의 맨 뒤에 붙이는 것이 중요했다.

처음에는 CMakeLists.txt 에서

```
set (CMAKE_C_FLAGS ...)
```

```
set (CMAKE_EXE_LINKER_FLAGS ...)
```

설정값만을 알고 있었기 때문에 이를 통해서 원하는 바를 구현하려고 했다.

그러나 매번

```
make VERBOSE=1
```

로 확인해볼 때마다 -pthread -lrt 가 컴파일 커맨드의 중간에 들어가 있어서 당황스러웠다.

나중에

```
target_link_libraries (
```

```
)
```

의 존재를 확인하고 이를 활용해

```
target_link_libraries ( ${MY_TARGET} pthread rt )
```

라인을 작성해 넣으므로써 문제없이 빌드 프로세스를 완료할 수 있었다.

위의 과정에서 겪은 어려움이 빌드 프로세스를 조금 더 명확하게 이해하는 계기가 되었다.

정적 / 동적 라이브러리는 링크 과정에서 애플리케이션에 연결하는 것이다.

[.c 텍스트] → “컴파일” → [.o 오브젝트 파일] → “링크” → [.out 실행파일]

4. Source Codes (with comments)

프로젝트 구조

```
$ tree .
```

```
.
├── 201202139_assignment_01_report.odt
├── CMakeLists.txt
├── include
│   ├── buffer.h
│   ├── consumer_semaphore.h
│   ├── print_functions.h
│   ├── producer_consumer_shared_memory_main.h
│   └── producer_semaphore.h
├── README.md
└── src
    ├── consumer_semaphore.c
    ├── print_functions.c
    ├── producer_consumer_shared_memory_main.c
    └── producer_semaphore.c
```

2 directories, 12 files

README.md

```
1 | Terminal Command Examples
2 |
3 | <BUILD>
4 |   (in the root directory of project)
5 | $ mkdir build && cd build && cmake ..
6 | $ make
7 | $ ./01_Producer_Consumer
8 |
9 | <Clear>
10 |   (in the build directory)
11 | $ cd .. && rm -rf build/
12 |
13 |
```

CMakeLists.txt

```
1 | cmake_minimum_required (VERSION 2.6)
2 |
3 | set (project_title "producer_consumer_shared_memory")
4 | set (project_author "yks93")
5 | set (project_revised_data "2018-10-16")
6 |
7 | set (INCLUDE_DIRS ${PROJECT_SOURCE_DIR}/include)
8 |
9 | project (${project_title})
10 | # The version number.
11 | set (${project_title}_VERSION_MAJOR 1)
12 | set (${project_title}_VERSION_MINOR 0)
13 | set (${project_title}_VERSION_PATCH 1)
14 |
15 | # language standard, compiler settings
16 | set (CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -Wall")
17 |
18 | # Bring the headers into the project
19 | include_directories(${INCLUDE_DIRS})
20 |
21 | set (SRCS
22 |     src/consumer_semaphore.c
23 |     src/producer_semaphore.c
24 |     src/print_functions.c
25 |     src/producer_consumer_shared_memory_main.c
26 | )
27 |
28 | add_executable (01_Producer_Consumer ${SRCS})
29 |
30 | # link with libraries
31 | target_link_libraries(01_Producer_Consumer
32 |     pthread
33 |     rt
34 | )
35 |
```

producer_consumer_shared_memory_main.c

```

1  /* ***** */
2  * FILE: producer_consumer_shared_memory_main.c
3  * TITLE: Main program,
4  * multi-processing producer-consumer C code adopting
5  * POSIX unnamed semaphore for synchronization
6  * of two processes.
7  * Using POSIX Shared Memory object for inter-process
8  * communication
9  * Date: October 13, 2018
10 * Revised: October 16, 2018
11 * Author: yks93
12 * Email: keesung0705@gmail.com
13 * Assignment: Operating Systems - Assignment #01
14 * Due Date: ~
15 *
16 * Problem Statement: Implement a simple program using POSIX Semaphore.
17 * Create POSIX Shared Memory between two processes to act
18 * as a buffer for data transfer. These two processes
19 * communicates in the producer-consumer mode.
20 * The producer process loops through 0 to 1000, transfers
21 * sequential integer values to the consumer through
22 * the shared memory.
23 * The consumer process loops through 0 to 1000, retrieves
24 * the data from buffer.
25 *
26 * Compile: Use the CMakeLists.txt file
27  /* ***** */
28
29  /* Include header files */
30  #include "../include/producer_consumer_shared_memory_main.h"
31  |
32  /* ***** */
33  * Name of Function: main
34  * Purpose: Fork the main process, use each as a producer, consumer
35  * respectively. Create a POSIX Shared Memory to inter
36  * communicate with each other.
37  * Use POSIX unnamed semaphore to synchronize two processes
38  * Parameters: ()
39  * Return Value: int
40  * Calls to: access, shm_open, ftruncate, mmap, fork, wait
41  * exit
42  * initiate_variables
43  * produce_data, consume_data
44  * munmap, close, shm_unlink
45  * printf, print_int, print_str
46  * Called from: --
47  *
48  * Method:
49  * 1. check whether the shared memory object exists
50  * if there is, remove it
51  * 2. create, open, map a new POSIX Shared Memroy object
52  * 3. fork a process
53  *
54  * In Parent Process
55  * a. P-Operation on semaphore var which indicates
56  * remaining buffer space
57  * b. enqueue new data into buffer in shared memory
58  * c. V-Operation on semaphore var which indicates
59  * number of data in buffer
60  *
61  * In Child Process
62  * a. P-Operation on semaphore var which indicates
63  * number of data in buffer
64  * b. dequeue oldest data from buffer in shared mememory
65  * c. V-Operation on semaphore var which indicates
66  * remaining buffer space
67  * --after processing data complete
68  * d. unmap virtual memory of child process from
69  * shared memory object
70  * e. close the file descriptor refering to shared
71  * memory object
72  * f. exit explicitly with success status
73  *
74  * *****

```

```

70  *          4. unmap virtual memory of parent process from
71  *          shared memory object
72  *          5. close the file descriptor refering to shared
73  *          memory object
74  *          6. remove the shared memory object from file system
75  *          7. exit(return) with sucess status
76  *****/
77  int main()
78  {
79      /*
80       *   Declaring vaiables differentiate between parent and child process
81       *   var "pid" stores process-ID
82       */
83      pid_t pid;
84
85      /*
86       *   shm_fd
87       *   -> file descriptor for POSIX Shared Memory object
88       *   i_trunc, i_unmap, i_close, i_unlink
89       *   -> vars for error check
90       *   ptr
91       *   -> pointer to access the shared memeory object
92       *   -> returned value from mmap( ) function on success
93       */
94      int shm_fd;
95      int i_trunc, i_wait, i_unmap, i_close, i_unlink;
96      void* ptr;
97
98      {
99          /*
100         *   check whether the shared memory object of the NAME exists
101         *   if yes
102         *   -> remove using unlink( ) function
103         *   if no
104         *   -> do nothing
105         */
106
107         const char* NAME = "/dev/shm/BUFFER";
108         if (access(NAME, F_OK) == 0)
109         {
110             i_unlink = shm_unlink("BUFFER");
111             if (i_unlink == -1) {
112                 perror("Initialization Failed\n");
113                 exit(EXIT_FAILURE);
114             }
115             print_str("Initialize OK");
116         }
117
118         /* create and open a new POSIX Shared Memeory object */
119         shm_fd = shm_open("BUFFER", O_CREAT|O_RDWR, 777);
120         if (shm_fd == -1) {
121             perror("Shared Memory Open Failed\n");
122             exit(EXIT_FAILURE);
123         }
124         print_str("Open OK");
125
126         /* set the size of shared memeory object */
127         i_trunc = ftruncate(shm_fd, sizeof(shared_buffer_struct));
128         if (i_trunc == -1) {
129             perror("ftruncate( ) Failed\n");
130             exit(EXIT_FAILURE);
131         }
132         print_str("Ftruncate OK");

```

```

134  /* map the shared memory object into the virtual address space of the calling process */
135  ptr = mmap(NULL, sizeof(shared_buffer_struct), PROT_READ|PROT_WRITE, MAP_SHARED, shm_fd, 0);
136  if (ptr == MAP_FAILED) {
137      perror("Memory Mapping Failed\n");
138      exit(EXIT_FAILURE);
139  }
140  print_str("mmap OK");
141
142  /* initialize the variables - in, out, buffer, 2 semaphore */
143  initiate_variables(ptr);
144
145  /* fork a process */
146  if ((pid = fork()) &lt; 0) {
147      perror("fork() Failed\n");
148      exit(EXIT_FAILURE);
149  }
150  else if (pid != 0)
151  { // parent : PRODUCER
152      print_str("parent begin !!!\n");
153      produce_data(ptr);
154  }
155  else
156  { // child : CONSUMER
157      print_str("child begin !!!\n");
158      consume_data(ptr);
159
160      /*
161       * The main process was forked previously,
162       * now two processes (parent and child) have its own
163       * memory space
164       *
165       * This is clean-up code only for child process
166       */
167      i_unmap = munmap(ptr, sizeof(shared_buffer_struct));
168      if (i_unmap == -1) {
169          perror("m-Unmapping in Child Failed\n");
170          exit(EXIT_FAILURE);
171      }
172      print_str("memory Unmap in Child OK");
173
174      i_close = close(shm_fd);
175      if (i_close == -1) {
176          perror("Close Failed in Child \n");
177          exit(EXIT_FAILURE);
178      }
179      print_str("Close in Child OK");
180
181      /* end of the child process */
182      exit(EXIT_SUCCESS);
183  }
184
185  /* wait until the child process to exit */
186  if ((i_wait = wait(0)) == -1) {
187      perror("ERROR: unexpected error occured on child process exit\n");
188      exit(EXIT_FAILURE);
189  }
190  print_str("child process exit OK!");
191
192  /*
193   * Clean-up code for parent process
194   */
195
196  /* unmap the shared memory object from the virtual address space of the
197  i_unmap = munmap(ptr, sizeof(shared_buffer_struct));
198  if (i_unmap == -1) {
199      perror("m-Unmapping Failed\n");
200      exit(EXIT_FAILURE);
201  }
202  print_str("memory Unmap OK");

```



```

204     /* close the file descriptor allocated by shm_open( ) */
205     i_close = close(shm_fd);
206     if (i_close == -1) {
207         perror("Close Failed\n");
208         exit(EXIT_FAILURE);
209     }
210     print_str("Close OK");
211
212     /* remove the shared memory object itself by name */
213     i_unlink = shm_unlink("BUFFER");
214     if (i_unlink == -1) {
215         perror("Unlink Failed\n");
216         exit(EXIT_FAILURE);
217     }
218     print_str("Unlink OK");
219
220     /* end of the parent process. end of main */
221     return 0;
222 }
223
224 /*****
225  * Name of Function:    initiate_variables
226  * Purpose:             initiate the variables resided inside the shared memory
227  *                     the only parameter is type (void*) which points to the
228  *                     shared memory object
229  * Parameters:          (void*)
230  * Return Value:        --
231  * Calls to:            sem_init, memset
232  * Called from:         main    (file: producer_consumer_shared_memory_main.c)
233  *
234  * Method:              1. reset var 'in',
235  *                     which is pointing the index for a new data
236  *                     to be produced and enqueued
237  *                     2. reset var 'out',
238  *                     which is pointing the first data to be consumed
239  *                     3. reset var 'buffer'
240  *                     whole contents inside buffer is reset to -1
241  *                     (sem_init function initiate "unnamed semaphore")
242  *                     4. init semaphore condition variable 'buffer_has_space'
243  *                     to 100 which indicates how many more data
244  *                     can be produced
245  *                     5. init semaphore condition variable 'buffer_has_data'
246  *                     to 0 which indicates how many data is awaiting
247  *                     to be consumed
248  *****/
249
250 void initiate_variables(void* ptr) {
251     shared_buffer_struct* stptr = (shared_buffer_struct*) ptr;
252
253     stptr->in = 0;
254     stptr->out = 0;
255     memset(stptr->buffer, -1, sizeof(int) * BUF_SZ);
256     sem_init(&(stptr->buffer_has_space), 1, BUF_SZ);
257     sem_init(&(stptr->buffer_has_data), 1, ZERO);
258 }

```

producer_consumer_shared_memory_main.h

```
1  /*****
2  * FILE:                producer_consumer_shared_memory_main.h
3  * Title:               Header file for .c file which contains main function
4  *                     of this assignment
5  * Date:                October 13, 2018
6  * Revised:            October 16, 2018
7  * Author:              yks93
8  * Email:               keesung0705@gmail.com
9  * Assignment:          Operating Systems - Assignment #01
10 * Due Date:            ~
11 *
12 * Struct List:         --
13 * Function List:        initiate_variables
14 *
15 *****/
16
17 #ifndef __PRODUCER_CONSUMER_SHARED_MEMORY_f5c3433e_7e85_4bf5_a3ec_ec6f5a5027f1_H__
18 #define __PRODUCER_CONSUMER_SHARED_MEMORY_f5c3433e_7e85_4bf5_a3ec_ec6f5a5027f1_H__
19
20 /* include header files */
21 #include "buffer.h"
22 #include "producer_semaphore.h"
23 #include "consumer_semaphore.h"
24 #include "print_functions.h"
25
26 #define LOOP_COUNT 1000
27
28 /* function prototypes */
29 void initiate_variables(void*);
30
31 #endif /* __PRODUCER_CONSUMER_SHARED_MEMORY_f5c3433e_7e85_4bf5_a3ec_ec6f5a5027f1_H__ */
32 |
```

buffer.h

```
1  /*****
2  *
3  * FILE:                buffer.h
4  * Title:               Header file for utility which can be used to store
5  *                     data inside shared memory
6  * Date:                October 13, 2018
7  * Revised:            October 16, 2018
8  * Author:              yks93
9  * Email:               keesung0705@gmail.com
10 * Assignment:          Operating Systems - Assignment #01
11 * Due Date:            ~
12 *
13 * Struct List:         shared_buffer_struct
14 *
15 * Function List:        --
16 *****/
17
18 #ifndef __BUFFER_c7325cdf_587f_49d4_9730_0ed81e427d7a_H__
19 #define __BUFFER_c7325cdf_587f_49d4_9730_0ed81e427d7a_H__
20
21 #include <stdio.h>
22 #include <stdlib.h>
23 #include <string.h>
24 #include <limits.h>
25 #include <semaphore.h>
26 #include <sys/stat.h>
27 #include <sys/mman.h>
28 #include <sys/wait.h>
29 #include <sys/types.h>
30 #include <fcntl.h>
31 #include <unistd.h>
32
33 #define BUF_SZ 100
34 #define ZERO 0
```

```

36 /*****
37 * Name of Stuct:      shared_buffer_struct
38 * Purpose:           struct to be used by user to locate inside
39 *                   POSIX Shared Memory area
40 * Used By:           produce_consumer_shared_memory.c
41 * Member Variables:  1. in      (type: int)
42 *                   index where the producer process use to enqueue
43 *                   new data into buffer[]
44 *                   2. out      (type: int)
45 *                   index where the consumer process use to dequeue
46 *                   first data from buffer[]
47 *                   3. buffer[] (type: int[])
48 *                   circular queue in which the data are stored
49 *
50 * << Semaphore >>
51 * P(S) or wait(S):
52 *   S = S - 1
53 *   If S < 0 Then:
54 *     block the calling process
55 *     add it to the wait queue of this semaphore
56 *
57 * V(S) or signal(S):
58 *   S = S + 1
59 *   If S <= 0 Then:
60 *     removev the first process from the wait queue
61 *     add it to the scheduling queue
62 *
63 * 4. buffer_has_space (type: sem_t)
64 *   semaphore condition variable
65 *   used to control the behavior of the producer process
66 *   initial value 100
67 *   it indicates how many more items the producer
68 *   can put into buffer
69 * 5. buffer_has_data (type: sem_t)
70 *   semaphore condition variable
71 *   used to control the behavior of the consumer process
72 *   initial value 0
73 *   it indicates how many more items the consumer
74 *   can retrieve from buffer and use it
75 *****/
76 typedef struct shared_buffer_struct
77 {
78     int in, out;
79     int buffer[BUF_SZ];
80     sem_t buffer_has_space, buffer_has_data;
81 } shared_buffer_struct;
82
83 #endif /* __BUFFER_c7325cdf_587f_49d4_9730_0ed81e427d7a_H__ */
84
85

```

producer_semaphore.h

```
1  /*****
2  * FILE:                producer_semaphore.h
3  * Title:               Header file for .c file which implements the
4  *                     behaviors of producers
5  * Date:                October 13, 2018
6  * Revised:             October 16, 2018
7  * Author:              yks93
8  * Email:               keesung0705@gmail.com
9  * Assignment:          Operating Systems - Assignment #01
10 * Due Date:            ~
11 *
12 * Struct List:         --
13 * Function List:       produce_data
14 *
15 *****/
16
17 #ifndef __PRODUCER_SEMAPHORE_95b1eb64_21ab_4c11_a89a_ac72706a815e_H__
18 #define __PRODUCER_SEMAPHORE_95b1eb64_21ab_4c11_a89a_ac72706a815e_H__
19
20 #include "buffer.h"
21 #include "producer_consumer_shared_memory_main.h"
22
23 void produce_data(void*);
24
25 #endif /* __PRODUCER_SEMAPHORE_95b1eb64_21ab_4c11_a89a_ac72706a815e_H__ */
26
```

producer_semaphore.c

```
1  /*****
2  * FILE:                producer_semaphore.c
3  * Title:               Source file which implements the behaviors of producer
4  * Date:                October 13, 2018
5  * Revised:             October 16, 2018
6  * Author:              yks93
7  * Email:               keesung0705@gmail.com
8  * Assignment:          Operating Systems - Assignment #01
9  * Due Date:            ~
10 *
11 * Function List:       produce_data
12 *
13 *****/
14
15 /* Include header files */
16 #include "../include/producer_semaphore.h"
17
18 /*****
19 * Name of Function:    produce_data
20 * Purpose:             produce and enqueue new data into the buffer at the
21 *                     shared memory
22 * Parameters:          (void*)
23 * Return Value:        --
24 * Calls to:            sem_wait, sem_post, printf
25 * Called from:         main    (file: producer_consumer_shared_memory_main.c)
26 *
27 * Method:              inside for Loop
28 *                     1. decrease the semaphore variable 'buffer_has_space'
29 *                       using sem_wait() function
30 *                     2. enqueue new data into the buffer
31 *                     3. move 'in' one step forward
32 *                     4. check whether producer cycled through the
33 *                       whole buffer
34 *                     5. increase the semaphore variable 'buffer_has_data'
35 *                       using sem_post() function
36 *                       so that consumer can work
37 *****/
38 void produce_data(void* ptr)
39 {
40     shared_buffer_struct* stptr = (shared_buffer_struct*) ptr;
41
42     int i;
43     for (i=0; i < LOOP_COUNT; ++i) {
44         sem_wait(&(stptr->buffer_has_space));
45         stptr->buffer[stptr->in] = i;
46         printf("inside parent: %3d\n", stptr->buffer[stptr->in]);
47         ++(stptr->in);
48         stptr->in %= 100;
49         sem_post(&(stptr->buffer_has_data));
50     }
51 }
```

consumer_semaphore.h

```
1  /*****
2  * FILE:                consumer_semaphore.h
3  * Title:               Header file for .c file which implements the
4  *                     behaviors of consumers
5  * Date:               October 13, 2018
6  * Revised:            October 16, 2018
7  * Author:             yks93
8  * Email:              keesung0705@gmail.com
9  * Assignment:         Operating Systems - Assignment #01
10 * Due Date:           ~
11 *
12 * Struct List:        --
13 * Function List:      consume_data
14 *
15 *****/
16
17 #ifndef __CONSUMER_SEMAPHORE_725010e0_90bd_4d8e_a278_6fd863b13e38_H__
18 #define __CONSUMER_SEMAPHORE_725010e0_90bd_4d8e_a278_6fd863b13e38_H__
19
20 #include "buffer.h"
21 #include "producer_consumer_shared_memory_main.h"
22
23 void consume_data(void*);
24
25 #endif /* __CONSUMER_SEMAPHORE_725010e0_90bd_4d8e_a278_6fd863b13e38_H__ */
26
```

consume_semaphore.c

```
1  /*****
2  * FILE:                consumer_semaphore.c
3  * Title:               Source file which implements the behaviors of consumer
4  * Date:               October 13, 2018
5  * Revised:            October 16, 2018
6  * Author:             yks93
7  * Email:              keesung0705@gmail.com
8  * Assignment:         Operating Systems - Assignment #01
9  * Due Date:           ~
10 *
11 * Function List:       produce_data
12 *
13 *****/
14
15 /* Include header files */
16 #include "../include/consumer_semaphore.h"
17
18 /*****
19 * Name of Function:    consume_data
20 * Purpose:             dequeue one data from the buffer at the shared memory
21 *                     and consume it
22 * Parameters:          (void*)
23 * Return Value:        --
24 * Calls to:            sem_wait, sem_post, printf
25 * Called from:         main (file: producer_consumer_shared_memory_main.c)
26 *
27 * Method:              inside for Loop
28 *                     1. decrease the semaphore variable 'buffer_has_data'
29 *                        using sem_wait() function
30 *                     2. dequeue the first data in the buffer
31 *                     3. move 'out' one step forward
32 *                     4. check whether consumer cycled through the
33 *                        whole buffer
34 *                     5. increase the semaphore variable 'buffer_has_space'
35 *                        using sem_post() function
36 *                        so that producer can work
37 *****/
38 void consume_data(void* ptr)
39 {
40     shared_buffer_struct* stptr = (shared_buffer_struct*) ptr;
41
42     int i;
43     for (i=0; i < LOOP_COUNT; ++i) {
44         sem_wait(&(stptr->buffer_has_data));
45         printf("\tchild: %3d\n", stptr->buffer[stptr->out]);
46         ++(stptr->out);
47         stptr->out %= 100;
48         sem_post(&(stptr->buffer_has_space));
49     }
50 }
```

print_functions.h

```
1  /*****
2  * FILE:                print_functions.h
3  * Title:               Header file for simplified implementations of printf
4  *                      functions
5  * Date:                October 14, 2018
6  * Revised:             October 16, 2018
7  * Author:              yks93
8  * Email:               keesung0705@gmail.com
9  * Assignment:          --
10 * Due Date:            ~
11 *
12 * Struct List:         --
13 *
14 * Function List:        printf
15 *                      print_str
16 *                      print_int
17 *
18 *****/
19
20 #ifndef __PRINT_FUNCTIONS_2483b38a_f255_452c_980f_9d100a03d610_H__
21 #define __PRINT_FUNCTIONS_2483b38a_f255_452c_980f_9d100a03d610_H__
22
23 #include <stdio.h>
24
25 void printf(void);
26 void print_str(const char *);
27 void print_int(int);
28
29 #endif /* __PRINT_FUNCTIONS_2483b38a_f255_452c_980f_9d100a03d610_H__ */
30
```

print_functions.c

```
1  #include "../include/print_functions.h"
2
3  /* prints only a new line */
4  void printf(void)
5  {
6      printf("\n");
7  }
8
9  void print_str(const char * string)
10 {
11     printf("%s\n", string);
12 }
13
14 void print_int(int i)
15 {
16     printf("%d\n", i);
17 }
18
19
20
21
```

4. Results Capture

File Edit View Search Terminal Help

yks93 ~/Documents/assignments/Operating_Systems
re/build ./01_Producer_Consumer

Open OK

Ftruncate OK

mmap OK

parent begin !!!

inside parent: 0
inside parent: 1
inside parent: 2
inside parent: 3
inside parent: 4
inside parent: 5
inside parent: 6
inside parent: 7
inside parent: 8
inside parent: 9
inside parent: 10
child begin !!!
inside parent: 11

inside parent: 12
inside parent: 13
inside parent: 14
inside parent: 15
inside parent: 16
inside parent: 17
inside parent: 18
inside parent: 19
inside parent: 20
inside parent: 21
inside parent: 22
inside parent: 23
child: 0
inside parent: 24
child: 1
inside parent: 25
child: 2
inside parent: 26
child: 3
inside parent: 27
child: 4
inside parent: 28
child: 5
inside parent: 29
child: 6
inside parent: 30
child: 7
inside parent: 31
child: 8
inside parent: 32
child: 9
inside parent: 33
child: 10
inside parent: 34
child: 11
inside parent: 35
child: 12

File Edit View Search Terminal Help

child: 963

inside parent: 987

child: 964

inside parent: 988

child: 965

inside parent: 989

child: 966

inside parent: 990

child: 967

inside parent: 991

child: 968

inside parent: 992

child: 969

inside parent: 993

child: 970

inside parent: 994

child: 971

inside parent: 995

child: 972

inside parent: 996

child: 973

inside parent: 997

child: 974

inside parent: 998

child: 975

inside parent: 999

child: 976

child: 977

child: 978

child: 979

child: 980

child: 981

child: 982

child: 983

child: 984

child: 985

child: 986

child: 987

child: 988

child: 989

child: 990

child: 991

child: 992

child: 993

child: 994

child: 995

child: 996

child: 997

child: 998

child: 999

memory Unmap in Child OK

Close in Child OK

child process exit OK!

memory Unmap OK

Close OK

Unlink OK

yks93 ~/Documents/assignments/Operating_Systems
re/build