Embedded System Software 과제 1

(과제 수행 결과 보고서)

과 목 명 : [CSE4116] 임베디드시스템소프트웨어

담당 교수 : 서강대학교 컴퓨터공학과 박성용

학번/이름 : 20151556 변홍수

개발 기간 : 2020. 04. 24. ~ 2020. 04. 29.

I. 개발 목표

Device Control과 IPC를 이용하여 주어진 Clock, Counter, Text editor, Draw board를 구현한다.

II. 개발 범위 및 내용

가. 개발 범위

1) Main Process

이 프로세스에서는 연산의 실질적인 수행을 한다. 입력 프로세스한테 값을 받아서, 각 모드에 따른 적절한 연산을 한다. 그 다음, 연산이 된 값을 IPC 를 통해 출력 프로세스(Output Process)로 전달하는 역할을 한다.

2) Input Process

이 프로세스는 계산을 위해 입력을 받는 입력 장치를 관리한다. 주기적으로 입력 장치로부터 새로 들어온 값이 있는지 확인하고, 새로 들어온 값을 Main Process로 IPC 를 사용하여 전달한다. 입력 장치는 BACK, VOL+, VOL- 버튼과 9개의 스위치 버튼을 사용한다.

3) Output Process

이 프로세스는 Main process로부터 전달 받은 값을 해당하는 장치에 출력 하는 역할을 한다. 본 프로그램의 IPC 는 Shared Memory 방법을 이용하여 구현한다. 출력장치로는 LED, FND, Text LCD, Dot Matrix를 사용한다.

4) Device Control

Device Control에는 mmap() 함수를 이용하는 방법과 디바이스 드라이버를 사용하는 방법을 사용한다. LED는 mmap() 함수를 사용 하고 나머지 디바이스들은 Device Driver를 사용하여 프로그램을 구현한다.

5) IPC(Shared Memory)

Process간 통신을 위해 Shared Memory를 사용한다. 3개의 Process가 Shared Memory에 동시에 접근할 수 없도록 Critical Section 처리를 위해 세마포어(Semaphore)를 구현하여 사용한다.

나. 개발 내용

4가지 모드의 기능을 구현한다.

모드1 : Clock, 모드2 : Counter, 모드3 : Text Editor, 모드4 : Draw Board

---

기본적으로 모드가 변경되면, 각 모드 의 초기 상태 가 된다. 또한 프로그램 시작 시 기본 모드(default mode)는 모드 1 이다. 각 모드 변경 시, 현재 사용 중인 모드를 제외한 다른 모드에서 사용하는 디바이스들은 모두 초기화 시킨다.

모드 변경은 VOL+ / VOL- 버튼을 사용하며, Back버튼 입력 시 모든 디바이스를 초기화하고 프로그램을 종료한다.

VOL+ / VOL- 버튼 변경 순서는 다음과 같다.

VOL+ : Clock → Counter → Text Editor → Draw Board → Clock → …

VOL- : Clock → Draw Board → Text Editor → Clock → …

각 모드의 입출력 방식은 다음과 같다.

1) Clock

- FND : 시간을 출력한다. (Format : [HHMM]) 초기 상태는 보드의 시간이다.
- LED : 초기 상태에는 1번 LED에만 불이 들어온다. 시간을 변경하기 시작하면 3, 4번 LED가 1초씩 번갈아 가며 불이 들어온다. 변경이 끝나면 다시 1번 LED에만 불이 들어온다.
- SW1 : 보드의 시간 변경/저장 Toggle 기능을 한다.
- SW2 : 변경 중인 시간을 초기화하여 원래 보드의 시간으로 Reset한다.
- SW3 : 1시간을 증가시킨다.
- SW4 : 1분을 증가시킨다.

✓ **시간 변경 중 저장하지 않고 모드를 변경할 시 보드에 저장되지 않는다**.

2) Counter

- FND : 초기 상태는 '0000'이다. 카운팅 된 숫자를 출력한다. 진법 변경 시 현재 카운팅 숫자를 변환된 진법으로 출력한다. 10진수는 끝 세 자리만 출력한다.
- LED : 초기상태에는 2번 LED에만 불이 들어온다. 각 LED는 진수를 나타낸다.
  1번 LED : 2진수, 2번 LED : 10진수, 3번 LED : 8진수, 4번 LED : 4진수
- SW1 : 진수 변환을 한다. 10진수 → 8진수 → 4진수 → 2진수 → 10진수 → …
- SW2 : 백의 자리 숫자를 1 증가시킨다.
- SW3 : 십의 자리 숫자를 1 증가시킨다.
- SW4 : 일의 자리 숫자를 1 증가시킨다.

3) Text Editor

- FND : 초기 상태는 '0000'이다. 현재 Text를 입력하기 위해 몇 번의 스위치를 눌렀는지 카운트 한 값을 출력한다 9999를 넘어갔을 경우 만 단위 이상은 생략한다.

  스위치는 알파벳과 숫자를 입력한다. 다음 그림은 각 스위치에 해당하는 알파벳이다.

(1) .QZ	(2) ABC	(3) DEF
(4) GHI	(5) JKL	(6) MNO
(7) PRS	(8) TUV	(9) WXY

  초기 상태는 알파벳 입력이다. 한번 눌렀던 버튼을 다시 누를 때마다 해당 알파벳을 입력 횟수에 맞게 변경한다. **3회 이상 눌렀을 시 다시 처음으로 되돌아간다.** 새로운 버튼의 입력이 들어오면 기존의 출력한 문자 바로 다음에 문자를 출력한다.

- SW2+3 : 입력하던 텍스트를 모두 지우고 초기 상태로 돌아간다.
- SW5+6 : 알파벳 ↔ 숫자 입력을 변경한다.
- SW8+9 : 한 칸 공백을 입력한다.
- Text LCD : 초기 상태는 빈 상태이다. 스위치를 통해 입력된 Text를 출력한다. LCD의 최대 출력 범위를 넘어가면 기존의 Text에 가장 앞에 있던 문자를 제거하고 한 칸씩 앞으로 밀고 나서 새로 들어온 Text를 출력한다.
- Dot Matix : 초기 상태는 'A'이다. 현재 입력 받는 Text가 알파벳인지 숫자인지 나타낸다. 알파벳일 경우 'A'를, 숫자일 경우 '1'을 출력한다.

✓ **알파벳 입력에서 같은 스위치를 3회 이상 눌렀을 시 다시 처음으로 되돌아간다.**
Ex) A → B → C → A → …

4) Draw Board

- FND : 초기 상태는 '0000'이다. **현재 그림을 그리기까지** 몇 번의 스위치를 눌렀는지 카운트 한 값을 출력한다 9999를 넘어갔을 경우 만 단위 이상은 생략한다.

  각 스위치는 아래와 같은 기능을 담당한다.

(1) reset	(2) ↑	(3) 커서
(4) ←	(5) 선택	(6) →
(7) clear	(8) ↓	(9) 반전

-
- SW2, 4, 6, 8 : 방향키로 사용하며 커서를 이동한다.
- SW1 : 현재 그리고 있는 그림을 지우고 초기상태로 돌아간다.
- SW3 : 커서 표시/숨기기 기능이다. 초기 상태는 맨 왼쪽 첫 줄에 깜빡이는 상태이다. (1초에 한 번)

- SW7 : 현재 그리고 있는 그림을 지운다. 커서의 위치와 표시등은 바뀌지 않는다.
- SW9 : 현재 그림을 반전시킨다.
- Dot Matrix : 그림을 출력한다. 초기 상태는 커서를 맨 왼쪽 첫 줄에 깜빡인다. (1초에 한 번)

✓ **SW5 : Dot를 찍는다. 이미 Dot가 찍혀 있을 시 다시 지운다.**

✓ **커서 이동 중 화면을 넘어갔을 시 반대편으로 이동한다.**

III. 추진 일정 및 개발 방법

가. 추진 일정

2020. 04. 24.

- 프로그램을 구현하기 위한 기본적인 구조 마련
  - ✓ Main Process에서 fork()를 이용해 Input Process, Output Process 생성
  - ✓ Shared Memory(IPC)를 사용하기 위한 Semaphore 구현
  - ✓ VOL+ / VOL- 버튼을 이용해 4개의 모드 전환 구현
  - ✓ 스위치 작동
- Clock 구현 시작
  - ✓ FND(mmap), LED(Device Driver) 작동 테스트

2020. 04. 25.

- Clock 구현
  - ✓ 스위치 버튼으로 시간 변경
  - ✓ LED 깜빡임

2020. 04. 26.

- Counter 구현
  - ✓ FND 출력

- ✓ LED 전환
- ✓ 진법 전환
- ✓ 스위치 버튼 숫자 증가

2020. 04. 27.

- Text Editor 구현
  - ✓ 알파벳 변경 / 입력
  - ✓ 알파벳 ↔ 숫자 전환 및 입력
  - ✓ 공백 입력
  - ✓ 리셋 기능
  - ✓ LCD 범위 초과 시 기능
- Draw Board 구현 시작
  - ✓ Dot 깜빡임
  - ✓ 커서 움직임
- Clock(Editing) / Draw Board 에서 모드 전환 시 오류 픽스

2020. 04. 28.

- Draw Board 구현
  - ✓ Dot 찍기
  - ✓ 커서 표시/숨기기
  - ✓ Reset / Clear
  - ✓ 그림 반전
  - ✓ FND Counter
  - ✓ 커서 움직임 기능 추가(화면 넘어갔을 시 반대로 이동)
- Clock 시간 저장 기능 픽스
- Back 버튼 입력 시 프로그램 정상 종료 구현

✓ Input Process, Output Process 정상 종료 후 Main Process 종료

✓ 모든 디바이스 초기화

2020. 04. 29.

- Document 작성
- 최종 테스트 및 마이너 오류 픽스

나. 개발 방법

1) Main Process

Main Process는 Input Process와 Output Process 총 2개의 Process를 fork()한다.

```
...
064 int main(void){
065 struct my_pid{
066 pid_t input;
067 pid_t output;
068 }pid;
069
070
079 pid.input = fork();
080 if(pid.input == 0){
081 /*********************************************************
082 * input process
083 * *******************************************************/
...
140 };
140 else{
141 pid.output = fork();
142 if(pid.output == 0){
143 /*********************************************************
144 * output process
145 * *******************************************************/
...
363 }
364 else{
365 /*********************************************************
366 * main process
367 * *******************************************************/
...
727 }
```

Code 1

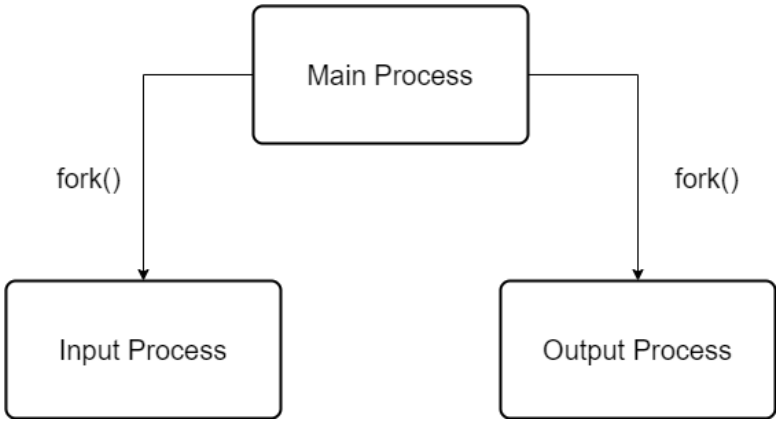

Figure 1

Main Process는 아래의 반복문을 종료 시까지 실행한다. Shared Memory *shamddr(shamddr[0])*을 통해 Input Process로부터 입력 값을 전달받는다. 입력 값이 종료 또는 모드 전환일 경우 적절한 연산을 수행한다. 그렇지 않은 경우 Shared Memory를 통해 현재의 모드를 Output Process에 전달하기 위해 *shamddr[0]*에 모드 값을 저장한다.

변수 *mode* 값 0, 1, 2, 3은 각각 모드 1, 2, 3, 4를 의미한다.

```
...
while(1){
input_button = shmaddr[0];

if(input_button == 158) { //Shutdown
semop (semid, &v3, NOPS);
semop (semid, &p1, NOPS);
return 0;
}

if(input_button == 114){ //vol down
mode += 3;
mode %= 4;
keep = 0;
input_button = 0;
}
else if(input_button == 115){ //vol up
mode++;
mode %= 4;
keep = 0;
input_button = 0;
}
shmaddr[0] = mode;
if(mode == 0){ //mode_1
...
}
else if(mode == 1){ //mode_2
...
}
else if(mode == 2){ //mode_3
...
}
```

```
604 else if(mode == 3){ //mode_4
...
726 }
...
```

Code 2

각 모드에서는 반복문 안에서 필요한 변수를 내부에 선언하여 사용한다. 아래는 각 모드에서 공통적으로 사용되는 변수와 반복문이 종료되어도 데이터가 필요한 변수들이다.

*boolean keep*은 *false*값으로 초기화 되어 있다. 모드에 진입 시 *true*, 변경 시 *false*값을 저장하여 모드의 변경을 확인하여 모드가 변경되었을 시 사용되는 디바이스 및 변수를 초기화 할 수 있다.

```
...
369 int mode = 0;
370 int input_button = 0;
371 int i;
372 bool keep = 0;
373 unsigned char fnd_data[4];
374
375 shmaddr = (int *) shmat(shmid, (char *) NULL, 0);
376 shmaddr[0] = 0; // init
377
378 //Nonvolatile variable
379 //Mode 1 First init
380 time_t now = time(NULL);
381 time_t time_tmp;
...
```

Code 3

- Clock

시간 변경 중 일 때 Output Process에서 LED가 깜빡이는 기능을 수행할 수 있게 *shammdr[1]*에 *MODE_CONTINUE*를 저장한다. 이때 *input_button*에 맞는 시간 변경 연산을 수행한다.

변경된 시간은 *time_tmp*에 저장한다. 변경 중 모드가 변경되는 등 비정상적으로 종료되었을 때는 변경 값을 보드 시간에 반영하지 않는다. 시간 변경이 정상적으로 종료 되었을 때 만 변경 값을 *now*에 반영한다.

FND 출력 값인 시간 데이터를 fnd_*data*에 저장하고 Shared Memeory *shmaddr[2]*를 통해 Output Process로 전달한다.

```
...
380 time_t now = time(NULL);
381 time_t time_tmp;
...
```

```
409 if(mode == 0){
410 /*********************************************************
411 * mode_1 variable
412 * ********************************************************/
413 struct tm *now_tm;
414 int hour, min;
415
416 if(keep == 0){
417 time_tmp = now;
418 shmaddr[1] = 0;
419 keep++;
420 }
421 if(input_button == 1){ //edit
422 if(shmaddr[1] == 0) { //start
423 shmaddr[1] = MODE_CONTINUE;
424 }
425 else { //end
426 now = time_tmp;
427 shmaddr[1] = 0;
428 }
429 }
430 else if(input_button == 2 && shmaddr[1] == MODE_CONTINUE){ //reset
431 time_tmp = now;
432 shmaddr[1] = 0;
433 }
434 else if(input_button == 4 && shmaddr[1] == MODE_CONTINUE){ //imp min
435 time_tmp += 60*60;
436 }
437 else if(input_button == 8 && shmaddr[1] == MODE_CONTINUE){ //imp hour
438 time_tmp += 60;
439 }
440 now_tm = localtime(&time_tmp);
441 hour = now_tm->tm_hour;
442 min = now_tm->tm_min;
443
444 fnd_data[0] = (unsigned char)hour/10;
445 fnd_data[1] = (unsigned char)hour%10;
446 fnd_data[2] = (unsigned char)min/10;
447 fnd_data[3] = (unsigned char)min%10;
448 memcpy(&(shmaddr[2]), fnd_data, sizeof(fnd_data));
449 }
...
```

Code 4

- Counter

입력 값 *Input_button*에 저장 된 값에 알맞은 연산을 수행한다. *shmaddr[1]*에 현재 진법을 저장하여 사용하고 Shared Memory를 통해 Output Process로 전달한다. *shmaddr[1]*의 값 0, 1, 2, 3은 각각 10, 8, 4, 2 진법을 의미한다.

*int sys[4]*에 순서대로 10, 8, 4, 2가 저장이 되어 sys접근을 이용해 진법 연산을 수행한다.

FND 출력 값을 fnd_*data*에 저장하고 Shared Memeory *shmaddr[2]*를 통해 Output

Process로 전달한다.

```
...
450 else if(mode == 1){
451 /*********************************************************
452 * mode_2 variable
453 * ********************************************************/
454 int num, num_tmp, sys_tmp;
455 int sys[4] = {10, 8, 4, 2};
456
457 if(keep == 0){
458 num = 0;
459 shmaddr[1] = 0;
460 keep++;
461 }
462 if(input_button == 1){
463 shmaddr[1]++;
464 shmaddr[1]%=4;
465 }
466 else if(input_button == 2){
467 num += sys[shmaddr[1]]*sys[shmaddr[1]];
468 }
469 else if(input_button == 4){
470 num += sys[shmaddr[1]];
471 }
472 else if(input_button == 8){
473 num++;
474 }
475
476 num_tmp = num;
477 sys_tmp = num_tmp%sys[shmaddr[1]];
478 fnd_data[3] = (unsigned char)sys_tmp;
479 num_tmp /= sys[shmaddr[1]];
480
481 sys_tmp = num_tmp%sys[shmaddr[1]];
482 fnd_data[2] = (unsigned char)sys_tmp;
483 num_tmp /= sys[shmaddr[1]];
484
485 sys_tmp = num_tmp%sys[shmaddr[1]];
486 fnd_data[1] = (unsigned char)sys_tmp;
487 num_tmp /= sys[shmaddr[1]];
488
489 if(sys[shmaddr[1]] == 10) fnd_data[0] = 0;
490 else{
491 sys_tmp = num_tmp%sys[shmaddr[1]];
492 fnd_data[0] = (unsigned char)sys_tmp;
493 num_tmp /= sys[shmaddr[1]];
494 }
495 memcpy(&(shmaddr[2]), fnd_data, sizeof(fnd_data));
496 }
...
```

Code 5

- Text Editor

입력 값 *Input_button*에 저장 된 값에 알맞은 연산을 수행한다. LCD 출력 값은

*lcd_string*에 저장한다. Text결정은 미리 선언된 *char text[4][9]*를 사용한다. 스위치 눌린 횟수로 1-row를 접근하고, 눌린 스위치의 번호로 2-row를 접근하여 Text를 결정한다. 알파벳과 숫자를 동시에 선언해 사용하기 위해 숫자 입력 시는 스위치가 눌린 횟수를 강제로 4로 고정시키는 방법을 사용한다.

FND 출력 Count값을 *fnd_data*에 저장되고 Shared Memeory *shmaddr[2]*를 통해, LCD 출력 lcd_string값을 Shared Memory *shmaddr[5]*를 통해 Output Process로 전달한다.

알파벳 입력 시 같은 스위치가 연속으로 여러 번 눌렸을 때 다른 알파벳을 출력해야 하기 때문에 *press_num*에는 같은 스위치가 연속으로 몇 번 눌렸는지 값이 저장된다. 또한 입력 모드에 따라 Dot Matrix 출력을 다르게 해야하기 때문에 *press_num*값을 Shared Memory *shmaddr[4]*를 통해 Output Process로 전달한다.

```
...
497 else if(mode == 2){
498 /*************************************************************
499 * mode_3 variable
500 * ************************************************************/
501 int count;
502 int press_num;
503 int input_button_prev;
504 int str_size;
505 unsigned char lcd_string[LCD_MAX_BUFF + 1];
506 char text[4][9] = {
507 {'.', 'A', 'D', 'G', 'J', 'M', 'P', 'T', 'W'},
508 {'Q', 'B', 'E', 'H', 'K', 'N', 'R', 'U', 'X'},
509 {'Z', 'C', 'F', 'I', 'L', 'O', 'S', 'V', 'Y'},
510 {'1', '2', '3', '4', '5', '6', '7', '8' ,'9'}
511 };
512 int find_input_num[9] = {1, 2, 4, 8, 16, 32, 64, 128, 256};
513 int input_num, num_tmp;
514
515 for(input_num = 0; input_num < 9; input_num++){
516 if(input_button == find_input_num[input_num]) break;
517 }
518
519 if(keep == 0){
520 press_num = 0;
521 count = -1;
522 str_size = 0;
523 memset(lcd_string, ' ', sizeof(lcd_string) - sizeof(char));
524 lcd_string[LCD_MAX_BUFF] = '\0';
525 input_button = input_button_prev = 0;
526 keep++;
527 goto space;
528 }
529
530 else if(input_button == 2+4){
531 press_num = 0;
532 count = -1;
533 str_size = 0;
534 memset(lcd_string, ' ', sizeof(lcd_string) - sizeof(char));
```

```
535 input_button = input_button_prev = 0;
536 goto space;
537 }
538 else if(input_button == 16+32){ //Change alpabet <-> number
539 if(press_num == 3) press_num = 0;
540 else press_num = 3;
541 }
542 else if(input_button == 128+256){ //Space
543 if(str_size == LCD_MAX_BUFF - 1){
544 for(i=0; i < LCD_MAX_BUFF - 1; i++)
545 lcd_string[i] = lcd_string[i+1];
546 str_size--;
547 }
548 if(str_size != 0) str_size++;
549 lcd_string[str_size] = ' ';
550 goto space;
551 }
552 else if(count == 0){
553 lcd_string[str_size] = text[press_num][input_num];
554 }
555 else if(press_num == 3){
556 if(str_size == LCD_MAX_BUFF - 1){
557 for(i=0; i < LCD_MAX_BUFF - 1; i++)
558 lcd_string[i] = lcd_string[i+1];
559 str_size--;
560 }
561 if(count == 1){
562 lcd_string[str_size] = text[press_num][input_num];
563 }
564 else{
565 lcd_string[++str_size] = text[press_num][input_num];
566 }
567 }
568 else if(input_button != input_button_prev) {
569 if(press_num != 3) press_num = 0;
570 if(str_size == LCD_MAX_BUFF - 1){
571 for(i=0; i < LCD_MAX_BUFF - 1; i++)
572 lcd_string[i] = lcd_string[i+1];
573 str_size--;
574 }
575 lcd_string[++str_size] = text[press_num][input_num];
576 }
577 else{
578 lcd_string[str_size] = text[press_num][input_num];
579 }
580 if(press_num != 3){
581 press_num++;
582 press_num%=3;
583 }
584 space:
585 input_button_prev = input_button;
586
587 count++;
588 num_tmp = count;
589 fnd_data[3] = (unsigned char)num_tmp%10;
590 num_tmp /= 10;
591
592 fnd_data[2] = (unsigned char)num_tmp%10;
593 num_tmp /= 10;
```

```
594
595 fnd_data[1] = (unsigned char)num_tmp%10;
596 num_tmp /= 10;
597
598 fnd_data[0] = (unsigned char)num_tmp%10;
599 num_tmp /= 10;
600 memcpy(&(shmaddr[2]), fnd_data, sizeof(fnd_data));
601 shmaddr[4] = press_num;
602 memcpy(&(shmaddr[5]), lcd_string, sizeof(lcd_string));
603 }
...
```

Code 6

- Draw Board

입력 값 *Input_button*에 저장 된 값에 알맞은 연산을 수행한다. Dot Matrix의 연산 값은 dot_edit에 저장하고 Shared Memory shmaddr[5]를 통해 Output Process로 전달한다

Output Process에서 Dot Matrix가 깜빡이는 기능을 수행할 수 있게 기본적으로 shammdr[1]에 *MODE_CONTINUE*가 저장된다.

*strcut cursor_point*에 커서의 위치가 저장되고 cursor_point.x, cursor_point.y각각 Shared Memory shmaddr[3], shmaddr[4]를 통해 Output Process로 전달한다.

FND 출력 *count*값을 *fnd_data*에 저장하고 Shared Memeory *shmaddr[2]*를 통해 Output Process로 전달한다.

```
...
604 else if(mode == 3){
605 /*********************************************************
606 * mode_4 variable
607 * ********************************************************/
608 struct cp{
609 int x;
610 int y;
611 }cursor_point;
612 unsigned char cursor_point_x_arr[7] = {64, 32, 16, 8, 4, 2, 1};
613 int j;
614 int count;
615 int num_tmp;
616
617 if(keep == 0){
618 cursor_point.x = 0;
619 cursor_point.y = 0;
620 for(i=0; i<10; i++){
621 dot_edit[i] = 0;
622 }
623 for(i=0; i<7; i++){
624 for(j=0; j<10; j++){
625 dot_chk[i][j] = 0;
626 }
627 }
```

```
628 count = -1;
629 shmaddr[1] = MODE_CONTINUE;
630 keep++;
631 }
632 if(input_button == 1){ //reset
633 cursor_point.x = 0;
634 cursor_point.y = 0;
635 for(i=0; i<10; i++){
636 dot_edit[i] = 0;
637 }
638 for(i=0; i<7; i++){
639 for(j=0; j<10; j++){
640 dot_chk[i][j] = 0;
641 }
642 }
643 count = -1;
644 shmaddr[1] = MODE_CONTINUE;
645 }
646 else if(input_button == 2){ //up
647 if(cursor_point.y > 0) cursor_point.y--;
648 else  cursor_point.y = 9;
649 }
650 else if(input_button == 8){ //left
651 if(cursor_point.x > 0) cursor_point.x--;
652 else cursor_point.x = 6;
653 }
654 else if(input_button == 32){ //right
655 if(cursor_point.x < 6) cursor_point.x++;
656 else cursor_point.x = 0;
657 }
658 else if(input_button == 128){ //down
659 if(cursor_point.y < 9) cursor_point.y++;
660 else cursor_point.y = 0;
661 }
662 else if(input_button == 4){ //cursor
663 if(shmaddr[1] == MODE_CONTINUE) shmaddr[1] = 0;
664 else shmaddr[1] = MODE_CONTINUE;
665 }
666
667 else if(input_button == 16){ //select
668 if(dot_chk[cursor_point.x][cursor_point.y]){
669 dot_chk[cursor_point.x][cursor_point.y] = 0;
670 dot_edit[cursor_point.y] -= cursor_point_x_arr[cursor_point.x];
671 }
672 else{
673 dot_chk[cursor_point.x][cursor_point.y] = 1;
674 dot_edit[cursor_point.y] += cursor_point_x_arr[cursor_point.x];
675 }
676 }
677
678 else if(input_button == 64){ //clear
679 for(i=0; i<10; i++){
680 dot_edit[i] = 0;
681 }
682 }
683
684 else if(input_button == 256){ //mirror
685 for(i = 0; i < 7; i++){
686 for(j = 0; j < 10; j++){
```

```
687 if(dot_chk[i][j] == 1){
688 dot_chk[i][j] = 0;
689 dot_edit[j] -= cursor_point_x_arr[i];
690 }
691 else{
692 dot_chk[i][j] = 1;
693 dot_edit[j] += cursor_point_x_arr[i];
694 }
695 }
696 }
697 }
698
699 if(input_button != 3) count++;
700
701 num_tmp = count;
702 fnd_data[3] = (unsigned char)num_tmp%10;
703 num_tmp /= 10;
704
705 fnd_data[2] = (unsigned char)num_tmp%10;
706 num_tmp /= 10;
707
708 fnd_data[1] = (unsigned char)num_tmp%10;
709 num_tmp /= 10;
710
711 fnd_data[0] = (unsigned char)num_tmp%10;
712 num_tmp /= 10;
713 memcpy(&(shmaddr[2]), fnd_data, sizeof(fnd_data));
714
715 shmaddr[3] = cursor_point.x;
716 shmaddr[4] = cursor_point.y;
717 shmaddr[5] = dot_chk[cursor_point.x][cursor_point.y];
718 memcpy(&(shmaddr[6]), dot_edit, sizeof(dot_edit));
719 }
...
```

Code 7

2) Input Process

입력 디바이스(VOL+, VOL-, Back, 스위치 버튼)를 오픈한다. Non Block으로 오픈하여 입력을 수시로 확인한다.

```
...
084 struct input_event ev[BUFF_SIZE];
085 int fd, rd, size = sizeof (struct input_event);
086
087 if((fd = open ("/dev/input/event0", O_RDONLY|O_NONBLOCK)) == -1) {
088 printf ("%s is not a vaild device.n", "/dev/input/event0");
089 }
091 int fd_s;
092 int buff_size_s;
093 unsigned char push_sw_buff[SWITCH_MAX_BUTTON];
094 if((fd_s = open ("/dev/fpga_push_switch", O_RDWR)) < 0) {
095 printf ("%s is not a vaild device.n", "/dev/fpga_push_switch");
096 }
097 buff_size_s = sizeof(push_sw_buff);
...
```

Code 8

아래의 반복문을 프로그램 종료 시까지 실행한다. VOL+, VOL- Back 버튼과 스위치 버튼을 구분하기 위해 변수 *switch_flag*를 사용한다.

스위치 입력 시 스위치가 동시에 여러 개 눌렸을 경우를 구분하기 위해 스위치 버튼의 인덱스를 2의 거듭제곱 값으로 넣어 그 값들의 합을 계산한다.

입력 값은 Shared Memory *shmaddr[0]*을 통해 Main Process의 *input_botton*으로 전달한다.

```
...
103 shmaddr = (int *) shmat(shmid, (char *) NULL, 0);
104 while (1){
105 usleep(SLEEP);
106
107 rd = read (fd, ev, (size_t)size * BUFF_SIZE);
108 read (fd_s, &push_sw_buff, (size_t)buff_size_s);
109
110 int i, j;
111 unsigned short int siwtch_flag = 0;
112 for(i = 0, j = 1; i < SWITCH_MAX_BUTTON; i++, j*=2) {
113 siwtch_flag |= push_sw_buff[i]*j;
114 }
115
116 if(siwtch_flag != 0){
117 shmaddr[0] = siwtch_flag;
118 //printf("wake p1\n");
119 semop (semid, &v1, NOPS);
120 semop (semid, &p2, NOPS);
121 }
122
123 if(rd > 0 && ev[0].value == 1){
124 shmaddr[0] = ev[0].code;
125 //printf("wake p1\n");
126 semop (semid, &v1, NOPS);
127
128 if(ev[0].code == 158) {
129 //shmaddr[1] = 0;
130 exit(0);
131 }
132 semop (semid, &p2, NOPS);
133 }
134 if(shmaddr[1] == MODE_CONTINUE){
135 semop (semid, &v1, NOPS);
136 semop (semid, &p2, NOPS);
137 }
...
```

Code 9

3) Output Process

출력 디바이스들을 오픈하고 초기화한다(코드 생략). 그 후 아래의 반복문을 프로그램 종료 시까지 실행한다.

Main Process에서 Shared Memory shamddr[0]을 통해 전달받은 mode 값에 알맞은 연산을 수행한다.

```
...
142 if(pid.output == 0){
143 /*********************************************************
144 * output process
145 * *******************************************************/
146 int mode;
147 unsigned char dot_set_blank[10] = {
148 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
149 };

...
222 while(1){
...
232 another_mode:
233 if(shmaddr[0] == 158){ //Shutdown
...
249 }
251 mode = shmaddr[0];
...
256 if(mode == 0){
...
290 }
291 else if(mode == 1){
...
297 }
298 else if(mode == 2){
...
309 }
310 else if(mode == 3){
...
327 }
328 else{
...
347 }
...
```

Code 10

- **Clock**

Main Process로부터 Shared Memory을 통해 전달받은 데이터 *fnd_data* 로 FND를 출력한다. LED 출력은 Output Process에서 결정한다.

LED가 깜빡이는 기능을 구현하기 위해 *shmaddr[1] = MODE_CONTIUE* 상태일 경우 mode 1의 반복문을 수행한다. Output Process가 잠들었다가 다시 살아나더라도 mode 1

이 지속될 수 있게 한다. 시간 변경이 종료되거나 모드가 변경되었을 경우 mode 1 반복문을 종료한다.

```
...
256 if(mode == 0){
257 memset(lcd_string, ' ', sizeof(lcd_string));
258 write(lcd_dev, lcd_string, LCD_MAX_BUFF);
259 if(shmaddr[1] == MODE_CONTINUE){
260 int k = 0;
261 while(1){
262 if(k <= 5){
263 *led_addr = 32;
264 }
265 else{
266 *led_addr = 16;
267 }
268 if(k%10 == 0) k = 0;
269
270 memcpy(fnd_data, &(shmaddr[2]), sizeof(fnd_data));
271 write(fnd_dev, fnd_data, 4);
272
273 semop (semid, &v2, NOPS);
274 semop (semid, &p3, NOPS);
275
276 //Mode Change
277 if(shmaddr[0] != mode){
278 shmaddr[1] = 0;
279 goto another_mode;
280 } // or Edit exit
281 else if(shmaddr[1] == 0) break;
282 k++;
283 }
284 }
285 *led_addr = 128;
286 memcpy(fnd_data, &(shmaddr[2]), sizeof(fnd_data));
287 write(fnd_dev, fnd_data, 4);
288
289 close(fnd_dev);
290 }
...
```

Code 11

- **Counter**

Main Process로부터 Shared Memory을 통해 전달받은 데이터로 LED와 FND를 출력한다. LCD는 사용하지 않으므로 Black 처리한다.

```
151 /*********************************************************
152 * mode_1 variable
153 * ********************************************************/
154 int fnd_dev;
155 unsigned char fnd_data[4];
156
157 int lcd_dev;
```

```
158 unsigned char lcd_string[32];
159
160 int led_dev;
161 unsigned long *fpga_addr = 0;
162 unsigned char *led_addr = 0;
...
291 else if(mode == 1){
292 memset(lcd_string, ' ', sizeof(lcd_string));
293 write(lcd_dev, lcd_string, LCD_MAX_BUFF);
294 *led_addr = (unsigned char)led[shmaddr[1]];
295 memcpy(fnd_data, &(shmaddr[2]), sizeof(fnd_data));
296 write(fnd_dev, fnd_data, 4);
297 }
...
```

Code 12

- **Text Editor**

Main Process로부터 Shared Memory을 통해 전달받은 데이터로 LED와 FND, LCD를 출력한다.

```
...
169 /*********************************************************
170 * mode_3 variable
171 * ********************************************************/
172 //int count;
173 int dot_dev;
174 size_t dot_str_size;
175
176 unsigned char fpga_dot[2][10] = {
177 {0x3E,0x7F,0x63,0x63,0x63,0x7F,0x7F,0x63,0x63,0x63}, // A
178 {0x0c,0x1c,0x1c,0x0c,0x0c,0x0c,0x0c,0x0c,0x0c,0x1e} // 1
179 };
180 int dot_flag_arr[4] = {0, 0, 0, 1};
181 int press_num;
182 bool dot_flag;
...
298 else if(mode == 2){
299 *led_addr = 0;
300 memcpy(fnd_data, &(shmaddr[2]), sizeof(fnd_data));
301 memcpy(lcd_string, &(shmaddr[5]), sizeof(lcd_string));
302 memcpy(fnd_data, &(shmaddr[2]), sizeof(fnd_data));
303 write(fnd_dev, fnd_data, 4);
304 write(lcd_dev, lcd_string, LCD_MAX_BUFF);
305 press_num = shmaddr[4];
306 dot_flag = dot_flag_arr[press_num];
307 dot_str_size = sizeof(fpga_dot[dot_flag]);
308 write(dot_dev, fpga_dot[dot_flag], dot_str_size);
309 }
...
```

Code 13

- **Draw Board**

Main Process로부터 Shared Memory을 통해 전달받은 데이터 *fnd_data*, *dot_edit*, *struct cursor_point* 로 FND와 Dot Matrix를 출력한다. *dot_chk*이 커서 깜빡임을 결정한다.

Clock과 마찬가지로 Dot가 깜빡이는 기능을 구현하기 위해 *shmaddr[1]* = MODE_CONTIUE 상태일 경우 mode 4의 반복문을 수행한다. Output Process가 잠들었다가 다시 살아나더라도 mode 4이 지속될 수 있게 한다. 모드가 변경되었을 때만 반복을 종료한다.

```
...
184 /*********************************************************
185 * mode_4 variable
186 * ********************************************************/
187 unsigned char dot_edit[10];
188 unsigned char cursor_point_x_arr[7] = {64, 32, 16, 8, 4, 2, 1};
189 struct cp{
190 int x;
191 int y;
192 }cursor_point;
...
310 else if(mode == 3){
311 *led_addr = 0;
312 if(shmaddr[1] == MODE_CONTINUE){
313 bool dot_chk;
314 int k = 0;
315 while(1){
316 memcpy(fnd_data, &(shmaddr[2]), sizeof(fnd_data));
317 write(fnd_dev, fnd_data, 4);
318
319 cursor_point.x = shmaddr[3];
320 cursor_point.y = shmaddr[4];
321 dot_chk = shmaddr[5];
322 memcpy(dot_edit, &(shmaddr[6]), sizeof(dot_edit));
323 if(dot_chk == 1){
324 if(k > 5){
325 dot_edit[cursor_point.y] -=
cursor_point_x_arr[cursor_point.x];
326 }
327 }
328 else{
329 if(k <= 5){
330 dot_edit[cursor_point.y] +=
cursor_point_x_arr[cursor_point.x];
331 }
332 }
333 if(k%10 == 0) k = 0;
334
335 write(dot_dev, dot_edit, sizeof(dot_edit));
336 semop (semid, &v2, NOPS);
337 semop (semid, &p3, NOPS);
```

```
338
339 //Mode Change
340 if(shmaddr[0] != mode){
341 shmaddr[1] = 0;
342 goto another_mode;
343 } // or Edit exit
344 else if(shmaddr[1] == 0) break;
345 k++;
346 }
347 }
348
349 memcpy(dot_edit, &(shmaddr[6]), sizeof(dot_edit));
350 memcpy(fnd_data, &(shmaddr[2]), sizeof(fnd_data));
351 write(fnd_dev, fnd_data, 4);
352 write(dot_dev, dot_edit, sizeof(dot_edit));
353 }
...
```

Code 14

4) Device Control

입출력에 필요한 디바이스를 제어하기 위해 미리 Define한 상수들은 다음과 같다.

```
...
021 #define BUFF_SIZE 64
022 #define KEY_RELEASE 0
023 #define KEY_PRESS 1
024 #define NOPS 1
025 #define SLEEP 200000
026 #define SWITCH_MAX_BUTTON 9
027
028 #define MAX_DIGIT 4
029 #define FND_DEVICE "/dev/fpga_fnd"
030
031 #define LCD_MAX_BUFF 32
032 #define LCD_LINE_BUFF 16
033 #define LCD_DEVICE "/dev/fpga_text_lcd"
034
035 #define FPGA_BASE_ADDRESS 0x08000000 //fpga_base address
036 #define LED_ADDR 0x16
037
038 #define DOT_DEVICE "/dev/fpga_dot"
...
```

Code 15

LED는 mmap()함수를 사용, 나머지 디바이스들은 Device Driver를 사용한다. LED의 경우 프로그램 종료 시 반드시 munmap()함수를 사용해 mapping한 메모리를 해제한다.

```
...
196 lcd_dev = open(LCD_DEVICE, O_WRONLY);
197 if (lcd_dev < 0) {
198 printf("Device open error : %s\n", LCD_DEVICE);
199 return -1;
```

```
200 }
201
202 led_dev = open("/dev/mem", O_RDWR | O_SYNC);
203 if (led_dev < 0) {
204 perror("/dev/mem open error");
205 return -1;
206 }
207
208 fpga_addr = (unsigned long *)mmap(NULL, 4096, PROT_READ | PROT_WRITE,
MAP_SHARED, led_dev, FPGA_BASE_ADDRESS);
209 if (fpga_addr == MAP_FAILED){
210 printf("mmap error!\n");
211 close(led_dev);
212 return -1;
213 }
214 led_addr = (unsigned char*)((void*)fpga_addr+LED_ADDR);
215
216 dot_dev = open(DOT_DEVICE, O_WRONLY);
217 if (dot_dev < 0) {
218 printf("Device open error : %s\n",DOT_DEVICE);
219 exit(1);
220 }
221
222 while(1){
223 fnd_dev = open(FND_DEVICE, O_RDWR);
224 if (fnd_dev < 0) {
225 printf("Device open error : %s\n", FND_DEVICE);
226 return -1;
227 }
...
358 munmap(led_addr, 4096);
...
```

5) IPC(Shared Memory)

Process간 통신 Shared Memory를 사용에서 Critical Section 처리를 위해 세마포어(Semaphore)를 구현하였다..

```
...
042 union semun {
043 int val;
044 struct semid_ds *buf;
045 unsigned short *array;
046 };
047
048 struct sembuf p1 = {0, -1, SEM_UNDO }, p2 = {1, -1, SEM_UNDO }, p3 =
{2, -1, SEM_UNDO };
049 struct sembuf v1 = {0, 1, SEM_UNDO }, v2 = {1, 1, SEM_UNDO }, v3 =
{2, 1, SEM_UNDO };
050
051 int getsem (void){
052 union semun x;
053 x.val = 0;
054 int id = -1;
055
```

```
056 id = (int)semget (IPC_PRIVATE, 3, 0600 | IPC_CREAT);
057 semctl ( id, 0, SETVAL, x);
058 semctl ( id, 1, SETVAL, x);
059 semctl ( id, 2, SETVAL, x);
060
061 return id;
062 }
...
```

기본적으로 Main Process → Output Process → Input Process → Main Process → … 순서를 반복한다. 각 Process에서는 예외적인 경우가 아닌 경우 반복문이 실행되어 잠들었다가 다시 깨어났을 때 이전과 동일한 작업을 수행할 수 있다.

Process Sequence는 다음과 같다.

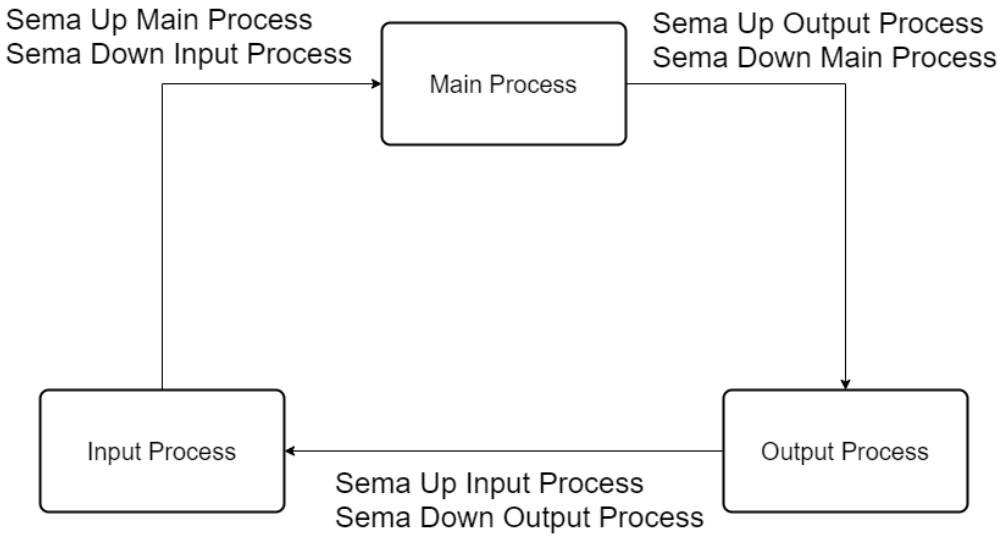

Figure 2

Input Process는 입력이 들어올 때까지 잠에 들지 않는다. 그러나 Output Process에서 LED나 Dot가 깜빡이는 기능을 구현하기 위해서 Input Process에서 *shamddr[1] = MODE_CONTINUE*인 경우 강제로 잠에 들게 하여 Process Sequence 진행시키면서 입력값을 확인함과 동시에 Output Process에서는 정해진 시간에 맞게 깜빡이는 기능을 수행할 수 있게 한다.

```
...
064 int main(void){
...
078 int semid = getsem();
...
080 if(pid.input == 0){
081 /*************************************************************
082 * input process
```

```
083 * ******************************************************/
...
104 while (1){
105 usleep(SLEEP);
123 if(rd > 0 && ev[0].value == 1){
126 semop (semid, &v1, NOPS);
...
132 semop (semid, &p2, NOPS);
133 }
134 if(shmaddr[1] == MODE_CONTINUE){
135 semop (semid, &v1, NOPS);
136 semop (semid, &p2, NOPS);
137 }
...
142 if(pid.output == 0){
143 /*********************************************************
144 * output process
145 * ******************************************************/
222 while(1){
229 semop (semid, &p3, NOPS);
...
356 semop (semid, &v2, NOPS);
...
364 else{
365 /*********************************************************
366 * main process
367 * ******************************************************/
...

387 while(1){
720 //printf("wake-sig p3\n");
721 semop (semid, &v3, NOPS);
722 //printf("sleep p1\n");
723 semop (semid, &p1, NOPS);
724 }
...
```

Code 16

IV. 연구 결과

- Main Process에서 2개의 Input Process, Output Process를 fork
- 3개의 프로세스간 Shared Memory(IPC)를 이용하여 프로세스간 통신
- Device Driver와 mmap를 이용한 디바이스 컨트롤
- 4가지의 모드를 구현하기 위해 Main Process, Output Process에서 4가지의 모드 처리
- 여러가지 상황에 대한 예외 처리

- 프로그램 전체 흐름도

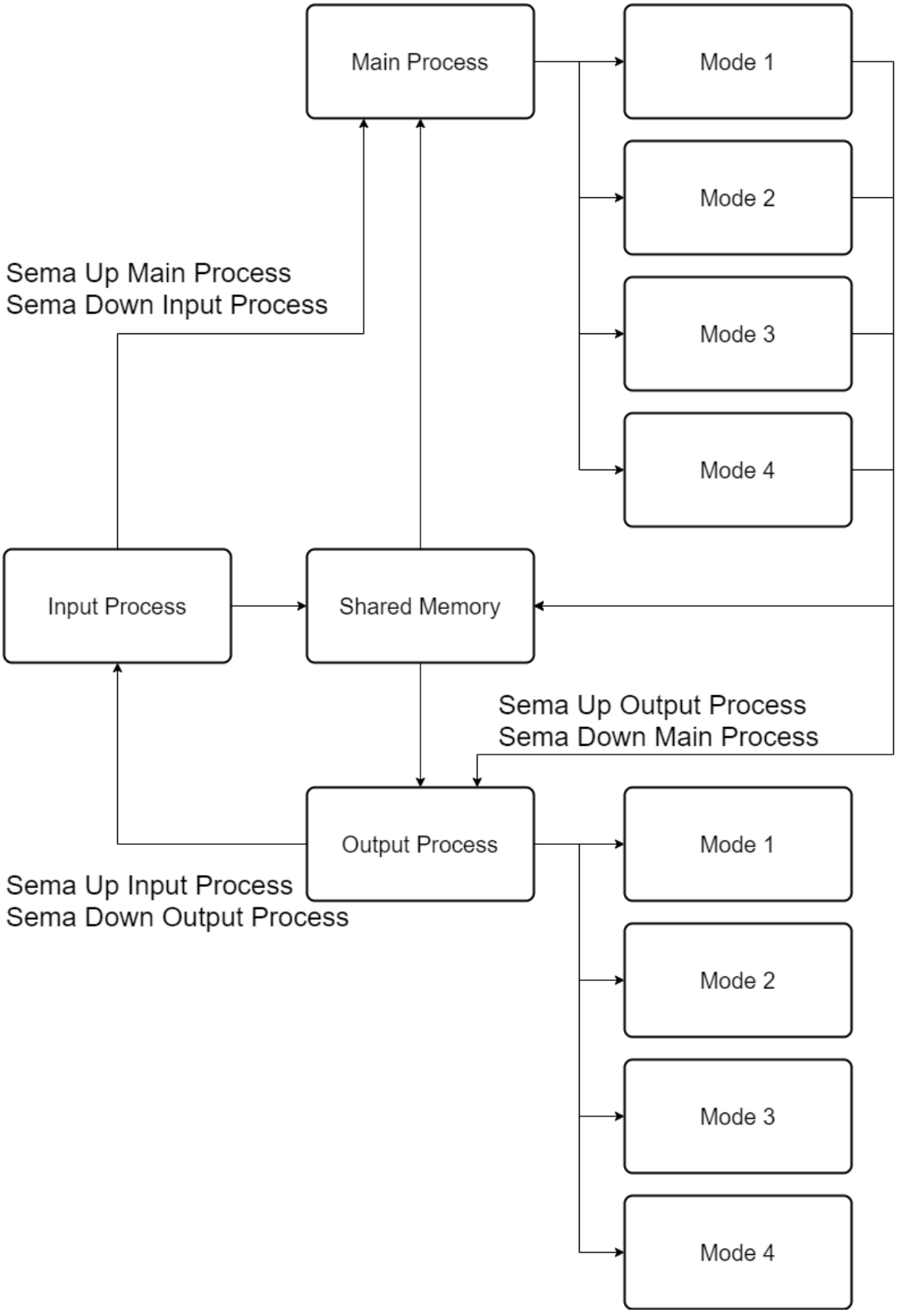

Figure 3

V. 기타

코드를 작성할 때 마다 보기 좋고 깔끔한 코드를 작성해야 하는지, 코드가 작동 시 효율이 좋게 작성할지 고민을 했다. 많은 경우 후자를 선택하게 되며 불필요한 함수 호출을 줄이기 위해 함수 작성을 최소화 하며 코드를 작성하게 되었는데 결국 작성된 코드를 보디 이도저도 아닌 코드가 된 것 같다.

시간적 여유가 있었더라면 조금 더 좋은 코드로 고칠 수 있지 않았을까 하는 아쉬움과 처음부터 그렇게 작성하지 못한 실력의 부족과 더 열심히 할 필요가 있음을 다시 한 번 느끼게 되었다.

Virtual Machine을 사용하여 Cross Compile을 통해 프로그램을 만드는 것도 새로운 경험이었다. Virtual Machine위에서의 개발 환경은 Unbuntu사용과는 별개로 Virtual Machine 이라는 기본적인 제한사항이 있어 약간의 불편함이 있었다. 이를 해소하고자 Virtual Machine과 Host PC와의 공유 폴더를 이용하여 실제 코드 작성은 기존과 같이 진행 할 수 있었고 Cross Compile만 Virtual Machine Ubuntu를 사용하는 방법을 사용하여 나름대로의 시간을 절약할 수 있었다.

-