2023 -1

**Embedded System Software**

Homework #1



컴퓨터공학과 응용빅데이터공학전공

120230200

김지섭

**IPC - Shared Memory**

**Processe간의 통신과정에서 shared momory를 사용하였습니다.**

**Get을 위한 key, Put을 위한 Key, value를 I/O device의 switch로 입력을 받기 때문에, 각 data를 공유하기 위한 통신, 그리고 data를 shared memory에 write, read 했다는 것을 각 process가 알 수 있도록 하기 위한 flag를 shared variable로 만들었습니다.**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**위의 이미지처럼 각 data 변수와 flag 변수를 shared memory를 얻습니다. 이때, key는 IPC\_PRIVATE으로 설정하면, 고유한 값을 자동으로 생성해줍니다. Size는 각 변수별로 SIZE값을 define하여 사용하였습니다. Shmflag는 새로운 segment를 만드는 것이기 때문에 IPC\_CREAT으로 설정하였습니다.**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**모든 shared variable에 대해 위의 이미지 처럼 attach를 하였습니다. shmaddr값은 NULL을 주어, kernel에서 자동으로 할당하도록 하였습니다.**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**이렇게 할당된 shared memory는 위의 코드와 같이, process간의 통신에 사용하였습니다**

**위의 코드는 readyKey에서 back이 입력된 경우, 모든 process가 종료 routine을 실행해야 하기 때문에 shared memory를 사용하여, 값을 전달하는 코드입니다.**

**I/O Process**

**I/O process의 경우, 여러 I/O Device를 동시에 contorl해야 하는 process입니다. 과제 명세서에 LED, Motor, 등의 I/O Device는 특정 시간마다 반복적으로 동작을 수행하기 때문에, 단일 작업으로 수행하는데 한계가 있다고 생각하였고, 이를 해결하기 위해 I/O device 중 일부 동작을 thread로 수행시켰습니다.**

**Thread로 정해진 동작을 수행하고, thread들은 data영역, 즉 전역변수를 공유하기 때문에 전역변수인 flag 변수를 tirger로 활용하여, thread의 동작을 contorl하였습니다.**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**해당 코드는 I/O device를 contorl하기 위한 각각의 thread를 띄우는 코드입니다.**

**LED, ReadyKey, Motor, RND, Switch(number pad), reset switch를 thread로 동작하도록 하였습니다.**

**LED**

**LED의 경우 mmap으로 구현하였습니다. Mmap으로 값을 변경하는 함수를 만들고, 해당 함수를 호출하는 controller 함수를 만들어 이를 thread로 동작시켰습니다.**

**led\_flag 변수를 활용하여, led의 동작을 제어하였고, led\_flag는 다른 device 에서 입력받은 값에 따라 변경하도록 구현하였습니다.**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**해당 이미지는 mmap 구현 코드입니다.**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**Device file인 /dev/mem을 open하는데, option으로 Read/Write가 모두 가능한 O\_RDWR로 열었고, 바로 값이 적용될 수 있도록 O\_SYNC를 추가하였습니다.**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**실습 예제를 참고하여, LED device의 memory address받아왔습니다.**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**해당 pointer 주소에 0~255까지의 값을 주면 각 비트에 해당하는 LED가 1인 경우 빛나고, 0인경우 빛나지 않게 됩니다. 이를 활용하여 LED를 control 할 수 있는 함수들을 만들고, 각 함수들을 실행시키기 위해 thread를 사용하였습니다. Thread는 led\_flag 값에 따라 각 함수들을 실행시키고, 각 함수들이 mmap 함수를 실행시켜, LED device를 Control하도록 구현하였습니다.**

****

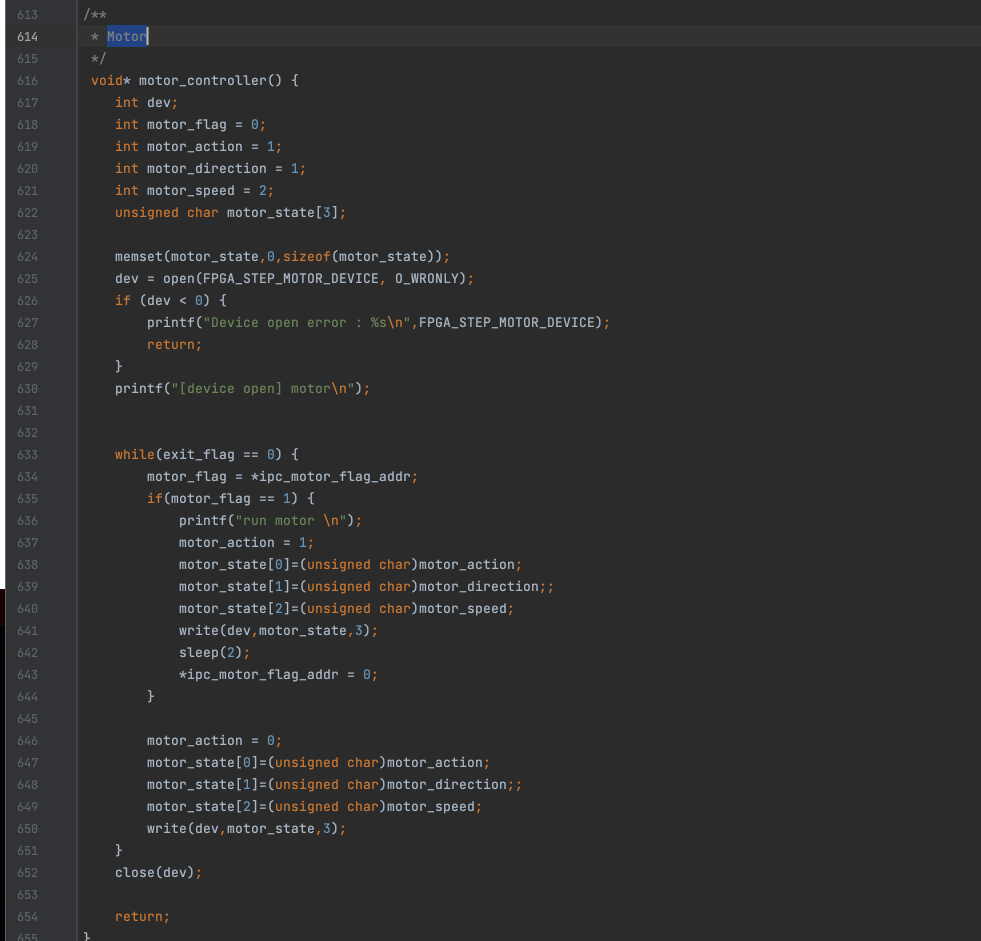
**해당 이미지와 같이, controller 함수를 thread로 동작시키고, flag에 따라 다른 함수를 실행하게 됩니다.**

**Motor**

**Motor는 merge를 하는 도중 1~2초 정도 구동되어야 했기 때문에 단일 process에서 sleep으로 구현하면, process가 sleep되어 있는 동안 나머지 다른 I/O devcie들에 대한 control이 어렵다는 문제가 있었습니다.**

**때문에 다른 I/O device 들과 같이 thread를 생성하여 동작하도록 구현하였습니다.**

**Motor는 merge 작업 중에 동작하기 때문에, I/O process가 아닌 merge process에서 모터의 동작 명령을 내려야 했고, shared memory로 ipc\_motor\_flag 변수를 만들어 control하였습니다.**

****

**FND**

**FND는 thread를 사용하여 지속적으로 fnd\_data array의 값을 확인후 띄워주는 방식으로 구현하였습니다.**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**Fnd의 예시에는 1000을 key로 입력하지만, switch에는 0을 입력할 수단이 없기 때문에 key 입력시, 가장 높은 자리부터 채우고, 입력되지 않은 값은 자동으로 0으로 처리하도록 구현하였습니다.**

**Number pad Switch**

**Switch는 지속적으로 입력을 check해야 하기때문에 역시 thread로 구현하였습니다. 같은 입력이라도 현재 상태가 영문인지, 숫자인지 현재 입력하고 있는 값이 key값인지 value인지 확인해야하기 때문에 여러 flag들을 사용하여 if문으로 이를 구분하였습니다.**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**우선 key 입력시에는 숫자만 입력이 가능하기 때문에 영문 상태라도, 숫자가 입력되게 하였고, 이를 인지할 수 있도록 warning message를 log로 출력하였습니다. 또한 fnd가 최대 4개의 숫자만을 띄울 수 있기 때문에 4개 이상의 입력은 처리하지 않았습니다.**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**영문 입력의 경우 buffer의 이전 값을 확인하고, 이전에 값을 입력하기 위해 누른 키와 현재 누른 키가 일치하는 경우, 이전에 입력한 값의 다음 알파벳을 넣는 방식으로 구현하였습니다.**

**Reset Switch**

**Reset switch는 단독으로 사용되는 의미보단, numberpad switch와 함께 사용되면서, key/value의 전환 기능을 하기 때문에 해당 키가 눌리는 경우, number pad switch에서 사용하는 flag를 변경하도록 구현하였습니다.**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**또한 이때, LED flag를 변경하여, 입력하는 값에 따라 반짝이는 Led를 변경하였습니다.**

**Text LCD**

**Text LCD는 thread를 사용하지 않고, 구현하였습니다. 이유는 다른 I/O device(switch), 다른 process(main)에서 값을 받아, 이를 보여주는 형태의 device이기 때문에, 다른 2가지 방식의 input의 혼동을 막기 위해서 control을 IO process와 switch\_controller에서 하도록 구현하였습니다.**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**위의 이미지와 같이, unsigned char array에 원하는 값을 넣어 Parameter로 넘겨 함수를 실행하는 방식으로 구현하였습니다.**

**Merge Process(Key Value Store)**

**Storage Table 저장 방법(Merge)**

**제가 구현한 KVS는 아래와 같은 방식으로 동작합니다.**

**[1]**

**도표이(가) 표시된 사진

자동 생성된 설명**

**우선 2개의 storage table이 가득 찼고, memory table에 3번째 key value data가 저장된 상황을 가정하겠습니다. 다음 상황에서는 mem table이 가득 찼기 때문에 flush가 발생하게 됩니다.**

**[2]**

**도표이(가) 표시된 사진

자동 생성된 설명**

**기존에 storage table이 2개였기 때문에, memory table의 데이터를 그대로 가지는 새로운 storage table인 3.st가 생성되게 됩니다.**

**[3]**

**도표이(가) 표시된 사진

자동 생성된 설명**

**총 storage table이 3개가 되면, 3개 중 오래된 2개의 storage table이 merge를 하게 됩니다. 현재 상황에서는 3.st가 막 생성되었기 때문에 1.st와 2.st가 merge하게 됩니다.**

**해당 과정에서 중복되는 data는 제거되고, index 및 각 data의 위치는 key값을 기준으로 정렬됩니다.**

**[4]**

**도표이(가) 표시된 사진

자동 생성된 설명**

**Merge 이후의 2개의 table입니다. 단순 생성 시간으로 따지면 3.st가 먼저 생성되었다고 할 수 있지만, 내부 데이터는 4.st가 먼저 write된 data이기 때문에, get이나 merge시에 4.st를 3.st보다 더 오래된 storage table로 처리합니다.**

**즉 다음 5.st가 생성되게 된다면, 4.st와 3.st가 merge되어 6.st가 생성됩니다.**

**또한 Get이 진행될때, 4.st보다 3.st를 먼저 search하게 됩니다.**