ku_ipc_lib.c

ku_msgget	Functionality	ioctl을 사용하여 메세지 큐 생성
	Parameters	int key:메세지 큐의 고유 식별자 int msgflg:IPC_CREAT,IPC_EXCL
	Return Value	msqid:성공,메세지 큐 식별 번호 return -1:실패

ku_ipc_lib.c

ku_msgsnd	Functionality	메세지 큐 식별자를 가지고 write()함수로 메세지 큐에 데이터 전달
	Parameters	-int msqid:큐 식별 번호 -void *msqp:큐에 전달 될 메세지 -int msgsz:메세지 사이즈 -msgflg IPC_NOWAIT if(msgflg & IPC_NOWAIT)가 0이 아니라면, 메시지는 전송되지 않고 즉시 return • if(msgflg & IPC_NOWAIT)가 0이라면, 메시지 큐가 비워지거나, 큐가 없어질 때까지 대기
	Return Value	0:성공 -1:실패

ku_ipc_lib.c

ku_msgrcv	Functionality	메세지 큐 식별자와 ,데이터 타입을 가지고 read()함수로 메세지 큐에서 메세지를 가져옴

	Parameters	-int msqid:큐 식별 번호
		-void *msqp:데이터를 받을 구조의 주소
		-int msgsz:받을 메세지 사이즈
		-long msgtyp:받을 메세지 타입
		-msgflg:
		IPC_NOWAIT : 메시지 큐에 메시지가 없을 때,
		if(msgflg & IPC_NOWAIT)가 0이 아니라면, 메시지는 수신되지 않고 즉시 return
		if(msgflg & IPC_NOWAIT)가 0이라면, 메시지 큐가 채워지거나, 큐가 없어질 때까지 대기
		MSG_NOERROR : 메시지 큐에 있는 자료가 msgsz보다 클 때,
		if(msgflg & MSG_NOERROR)가 0이 아니라면, 초과되는 메시지 자르고 읽을 수 있는 만큼만 수신
		if(msgflg & MSG_NOERROR)가 0이라면, 메시지 큐에 자료가 있어도 -1 return

ku_ipc_lib.c

	Return Value	rtn>0:받기 성공된 바이트 수 -1:실패

ku_ipc_lib.c

ku_msgclose	Functionality	msgqid로 명시된 메세지 큐 연결 해제
	Parameters	-int msqid:큐 식별 번호
	Return Value	0:성공 -1:실패

ku_ipc.c

open_interface	Functionality	캐릭터 디바이스 open시 호출
	Parameters	-struct inode:inode의 값 -struct file :파일 디렉토리
	Return Value	0:성공

ku_ipc.c

static long sys_ioctl	Functionality	-if(cmd==1) 메세지 큐 식별자로 받은 것 중 이미 할당된 큐 식별자가 있는지 확인 -if(cmd == 2) 메세지큐 할당 및 Linked list 에 추가 -if(cmd==3) 해당 큐 식별자를 가지고 메세지큐 삭제 만약 큐가 삭제 되면 그 해당하는 큐 구조체 정보에 있는 flag 값을 RCU를 이용해 바꾸고 해당 mqid에 해당하는 wait queue에 있는 리스트를을 다 깨운 후 에러 리턴을 할 수 있도록 한다.
	Parameters	-struct file :파일 디렉토리 -unsigned int cmd:ioctl 동작 구분자 -unsigned long : ioctl동작 시 필요한 데이터

ku_ipc.c

	Return Value	-if(cmd==1) 0:이미 해당 식별자 큐 존재 1:해당 식별자 큐 미존재 -if(cmd == 2) 1:메세지 큐 공간 할당 성공 -if(cmd==3) 0:큐 삭제 성공 -1:큐 삭제 실패

ku_ipc.c

read_msg	Functionality	유져 영역의 메세지를 커널에 존재하는 메세지큐로 전달한다.만약 wait 상태가 필요하다면 wait queue에 추가 되고 메세지가 읽어지면 순차적으로 메세지를 Size를 확인후 메세지 큐에 추가해 준다.
	Parameters	-struct file :파일 디렉토리 -char *buf:메세지 정보와 메세지 데이터가 담기는 구조체를 인자로 받는 포인터 변수 -size_t len:buf의 사이즈 -loff_t *lot
	Return Value	rtn>0:read 성공한 데이터 수 -1:큐의 용량이 가득 찼을때 -2:큐가 없을때 -3:에러

write_msg	Functionality	커널 영역에 있는 메세지 중에서 메세지 타입에 맞는 것 을 찾아 유저 영역으로 복사해 준다. 만약 wait상태가 필요하다면 wait queue에 추가를 하고 메세지가 send되면 wake up이 되어서 자신이 읽을 수 있는 메세지 인지 확인을 한다.
	Parameters	-struct file :파일 디렉토리 -char *buf:메세지 정보와 메세지 데이터가 담기는 구조체를 인자로 받는 포인터 변수 -size_t len:buf의 사이즈 -loff_t *lot
	Return Value	1:성공 -1:해당 타입의 메세지가 없을 때 -2:해당 큐가 없을 때 -3:에러

ku_ipc.c

ku_sys_v_init	Functionality	큐를 관리하는 Linked list 초기화 및 디바이스 등록
	Parameters	void
	Return Value	0:성공

ku_ipc.c

exit	Functionality	디바이스 해제
	Parameters	void
	Return Value	0:성공

Assignment 1과 다른점

1.Block I/O로 구현하여 Spinlock이 실행되는 것을 줄임

2.처리 순서를 FIFO구조로 하기 위해서 wait queue에 있는 메세지 싸이즈를 Linked list 로 저장하여 무조건 wake_up 시키지 않고 메세지 큐 싸이즈에 적합할때 일어나게함

3.FIFO구조를 통해 기아 현상이 일어나지 않게함
(만약 send 같은 경우 적합한 싸이즈 먼저 서비스를 처리하면 큰 싸이즈 메세지는 기아현상이 발생할 수 있음)

4.만약 큰 싸이즈의 메세지가 receive되면 send wait queue에서 메세지 큐 싸이즈에 맞게 계속해서 wake up을 시켜줌(좀 더 효율적 처리를 위하여)

5. receive 의 경우 메세지 큐에 받을 메세지가 send되면 타입 확인 후 타입이 맞지 않으면 return -1 한다. (메세지 큐에 들어갈 기회를 줬기 때문에 에러로 처리)

6.close실행을 하면 메세지 큐를 삭제하기 전에 RCU 와 flag 변수를 통해서 그 mqid에 해당하는 모든 wait상태의 메세지 처리를 에러 처리하고 삭제한다.

(flag를 이용할 경우 read write 문제에 좀 더 적합하다 생각해 RCU를 씀)

해결하지 못한 점

1.RCU를 써보니 레이스 컨디션 때문에 close함수가 재대로 작동하지 못한 점.
(여러 프로세스가 공용자원에 달려들면 synchronize의 처리를 하는 것이 복잡해 지는 것을 느낌)

느낀점

아직까지는 thread처리를 하지 않고 테스트를 해봐서 spinlock과 block I/O의 큰 속도 차이를 느끼지 못했지만 while문을 계속해서 실행하는 spinlock과 block I/O를 하는 것은 코드 flow 상에서도 좀 더 효율 적일 것 같다는 생각을 했습니다.