Lab -4

Bonus Question:

Design and implementation- I have used entries in the process table, to know which parameter is being passed that is ARECV or AALRM, and depending on that I take appropriate action. If its ARECV, it just behaves like registercb. So after the context switch code, if a process has a message then its callback function is called. I have written this code after context switch because, each time the receiver process is context switched and it becomes the current process, we need to check if it has received a message and if so we call the callback function. So similarly, if we pass the AALRM as an argument, the process must wait for a certain time, before its callback function is called, irrespective of whether or not it has the message.

I created the following test cases to check whether when we pass ARECV it gives the same output as in Part-2. And I got the same output as in Part2.

Test Case-1

Here there are two receiver processes and 1 send process. In the send process, I am sending messages to the two rececier process.

```
int bonusrecvhandler1(void){
  umsg32 msg;
    msg = receive();
    kprintf("Bonus1: msg received= %d\n",msg);
    return OK;
       }

int bonusrecvhandler2(void){
   umsg32 msg;
   msg = receive();
```

```
   kprintf("Bonus2: msg received= %d\n",msg);
   return OK;
}


void bonusrecv1(void)
{
  if (registercbsig(ARECV,&bonusrecvhandler1,0) != OK) {
         kprintf("recv1 handler registration failed\n");
        }

      while(1) {}
 }

void bonusrecv2(void)
 {
   if (registercbsig(ARECV,&bonusrecvhandler2,0) != OK) {
      kprintf("recv1 handler registration failed\n");
    }
    while(1) {}
 }

void send1(pid32 pid1,pid32 pid2 umsg32 msg1, umsg32 msg2)
```

```
{
  send(pid1, msg1);

  send(pid2, msg2);
}

pid32 rec1, rec2, snd1;
  rec1 = create(bonusrecv1,1024,20,"r1",0);
  rec2 = create(bonusrecv2,1024,20,"r2",0);
  snd1 = create(send1,1024,20,"s1",4,rec1,rec2,'a','b');
  resume(snd1);
  resume(rec1);
  resume(rec2);
```

Output:

Bonus1: msg received= 97

Bonus2: msg received= 98

Test Case : ARECV :2

Here I am creating one sender process and one receiver process. The sender process sends a message to the receiver, sleeps for sometime and sends another message to the same process. Since there is a sleep, the message will be consumed by the receiver then and it can accept the second message.

```
int bonusrecvhandler1(void){

    umsg32 msg;
    msg = receive();

    kprintf("Bonus1: msg received= %d\n",msg);
    return OK;
}



void bonusrecv1(void)
{
   if (registercbsig(ARECV,&bonusrecvhandler1,0) != OK) {
           kprintf("recv1 handler registration failed\n");
         }

       while(1) {}
 }

void send1(pid32 pid1, umsg32 msg1, umsg32 msg2)
{
   send(pid1, msg1);
   sleepms(5);
   send(pid1, msg2);
```

```
}

pid32 rec1, rec2, snd1;
  rec1 = create(bonusrecv1,1024,20,"r1",0);
snd1 = create(send1,1024,20,"s1",3,rec1,'a','b');
  resume(snd1);
  resume(rec1);
```

Output:

Bonus1: msg received= 97

Bonus1: msg received= 98

Test Cases-ARECV- 3

Here I am creating one sender process and one receiver process. The sender process sends a message to the receiver, and then immeadiatey sends another message to the same process. Since ,the message is being consumed by the receiver, it will not be ready to accept the second message and as a result that message is not printed.

```
int bonusrecvhandler1(void){

    umsg32 msg;
    msg = receive();

    kprintf("Bonus1: msg received= %d\n",msg);
    return OK;
}
```

```
void bonusrecv1(void)
{
  if (registercbsig(ARECV,&bonusrecvhandler1,0) != OK) {
          kprintf("recv1 handler registration failed\n");
        }

      while(1) {}
 }
void send1(pid32 pid1, umsg32 msg1, umsg32 msg2)
{
  send(pid1, msg1);
  send(pid1, msg2);
}

pid32 rec1, rec2, snd1;
 rec1 = create(bonusrecv1,1024,20,"r1",0);
snd1 = create(send1,1024,20,"s1",3,rec1,'a','b');
 resume(snd1);
 resume(rec1);
```

Output:

Bonus1: msg received= 97

Test cases-ALARM

Here I have created to 2 receiver process, and each of them have registered with the registercbsig. One of them is given a time of 50 ms before which its callback function is called and another one is given a time of 100ms. The following output observed tells us that the callback functions are called at the right time after the correct amount of time has passed.

```
int bonusrecvhandler3(void){
	kprintf("after 4 = %d\n", myglobalclock);
	kprintf("Bonus Passed\n");
	return OK;
}
int bonusrecvhandler4(void){
	kprintf("After-4 = %d\n", myglobalclock);

	 kprintf("Bonus Passed\n");
	return OK;
}

void bonusrecv3(void)
{

	kprintf("Before-3  = %d\n", myglobalclock);
	if(registercbsig(AALRM,&bonusrecvhandler3,50) != OK) {
```

```
            kprintf("recv1 handler registration failed\n");
       }
     while(1) {}
}

void bonusrecv4(void)
{

       kprintf("Before-4  = %d\n", myglobalclock);
       if(registercbsig(AALRM,&bonusrecvhandler4,100) != OK) {
            kprintf("recv1 handler registration failed\n");
}
    while(1){}
}

pid2 = create(bonusrecv3,1024,20,"r2",0);
 pid3 = create(bonusrecv4,1024,20,"r2",0);

 resume(pid2);
 resume(pid3);
```

Output:

Before-3 = 37

Before-4 = 41

after 3 = 91

Bonus Passed

After-4 = 145

Bonus Passed

Probelem 2:

Design and Implemenation: For each process, there is a pointer associated which points to the buffer area with which the receiver process has registered to. This is done by adding the entry abuf in the process table entry. Similarly, there is a pointer which points to the callback function. This again for each, I added an entry in the process table. IF a process registers a callback, using the system call, then these two entries of the process table are set i.e the the pointer to the callback function and the pointer to the buffer. So whenever this receiver process becomes the current running process, I am checking whether a callback function has been registered to that particular process. [This is done by checking whether the pointer to the callback function is NULL or not.] And apart from this, if there is a valid message for that particular process, I am then calling the callback function and also make sure to turn the flag prhasmsg = 0, since the process has consumed the message and the message is no longer valid. Also I am copying the the message to the buffer here.

The rationale behind choosing this approach is that it is very simple and its performance is also pretty good. And moreover, this approach ensures isolation/protection since it is made sure that the callback function is executed in the context of the function which registered the callback function.

Following are the test cases executed by me for this question.

Test Case1

Here there are two receiver processes and 1 send process. In the send process, I am sending messages to the two receiver process. The output is correct and also validates the design of Problem2. This is because the message sent to the receiver process is being delivered to it. And this is also done asynchronously. That is first the send process is resumed, it sends the message to the two different receivers. The two receivers when they are context switched in will see that they have a

valid message and since they also have registered for the callback , their callback is executed which prints the message received.

```
umsg32 proc1buf;
umsg32 proc2buf;

int myrecvhandler1(void) {
     kprintf("msg received = %d for process = %d\n",proc1buf,currpid);
     return(OK);
}

int myrecvhandler2(void) {

kprintf("msg received = %d for process = %d\n",proc2buf,currpid);
return(OK);
}

void recv1(void)
{
  if (registercb(&proc1buf,&myrecvhandler1) != OK) {
          kprintf("recv1 handler registration failed\n");
   }
   while(1) {}
}
```

```
void recv2(void)
{
  if (registercb(&proc2buf,&myrecvhandler2) != OK) {
        kprintf("recv2 handler registration failed\n");
 }
        while(1) {}
 }

void send1(pid32 pid1, pid32 pid2, umsg32 msg1, umsg32 msg2)
{

   send(pid1, msg1);
    send(pid2, msg2);
     }

pid32 rec1, rec2, snd1;
       rec1 = create(recv1,1024,20,"r1",0);
       rec2 = create(recv2,1024,20,"r2",0);
       snd1 = create(send1,1024,20,"s1",4,rec1,rec2,'a','b');
       resume(snd1);
       resume(rec1);
       resume(rec2);
```

Output:

msg received = 97 for process = 3

msg received = 98 for process = 4

Test Case 2:

Here I am creating one sender process and one receiver process. The sender process sends a message to the receiver, sleeps for sometime and sends another message to the same process. Since there is a sleep, the message will be consumed by the receiver then and it can accept the second message

```
int myrecvhandler1(void) {
     kprintf("msg received = %d for process = %d\n",proc1buf,currpid);
     return(OK);
}
void recv1(void)
{
  if (registercb(&proc1buf,&myrecvhandler1) != OK) {
         kprintf("recv1 handler registration failed\n");
   }
   while(1) {}
}

void send1(pid32 pid1, umsg32 msg1, umsg32 msg2)
{
  send(pid1, msg1);
```

```
  sleepms(5);
   send(pid1, msg2);
    }
pid32 rec1, rec2, snd1;
      rec1 = create(recv1,1024,20,"r1",0);

      snd1 = create(send1,1024,20,"s1",3,rec1,'a','b');
      resume(snd1);
      resume(rec1);
msg received = 97 for process = 3
msg received = 98 for process = 3
```

Test Case 3:

Here I am creating one sender process and one receiver process. The sender process sends a message to the receiver, and then immeadiatey sends another message to the same process. Since ,the message is being consumed by the receiver, it will not be ready to accept the second message and as a result that message is not printed.

```
int myrecvhandler1(void) {
     kprintf("msg received = %d for process = %d\n",proc1buf,currpid);
     return(OK);
}
void recv1(void)
{
  if (registercb(&proc1buf,&myrecvhandler1) != OK) {
```

```
        kprintf("recv1 handler registration failed\n");
    }
    while(1) {}
}

void send1(pid32 pid1, umsg32 msg1, umsg32 msg2)
{
  send(pid1, msg1);
  send(pid1, msg2);
   }
pid32 rec1, rec2, snd1;
      rec1 = create(recv1,1024,20,"r1",0);

      snd1 = create(send1,1024,20,"s1",3,rec1,'a','b');
      resume(snd1);
      resume(rec1);
```

Output:

msg received = 97 for process = 3

Test Case 4:

```
int myrecvhandler1(void) {
```

```
    kprintf("msg received = %d for process = %d\n",proc1buf,currpid);
    return(OK);
}

int myrecvhandler2(void) {

kprintf("msg received = %d for process = %d\n",proc2buf,currpid);
return(OK);
}

void recv1(void)
{
  if (registercb(&proc1buf,&myrecvhandler1) != OK) {
        kprintf("recv1 handler registration failed\n");
   }
   while(1) {}
}
void recv2(void)
{
 if (registercb(&proc2buf,&myrecvhandler2) != OK) {
       kprintf("recv2 handler registration failed\n");
 }
       while(1) {}
 }
```

```
void send1(pid32 pid1, umsg32 msg1)
{

  send(pid1, msg1);

 }

void send2(pid32 pid1, umsg32 msg1)
{
  send(pid1, msg1);
  }

pid32 rec1, rec2, snd1,snd2;
        rec1 = create(recv1,1024,20,"r1",0);
        rec2 = create(recv2,1024,20,"r2",0);
        snd1 = create(send1,1024,20,"s1",3,rec1,1);
        snd2 = create(send2,1024,20,"s2",3,rec2,2);
        resume(snd1);
        resume(rec1);
        resume(rec2);
        resume(snd2);
```

Output:

msg received = 1 for process = 3

msg received = 2 for process = 4

Problem 1:

Sendb – First I check the maxwait. IF its less than 0 it returns ERROR. If thesender is trying to send a message to the receiver and the receiver already has a message , then this sending process must be blocked. For this I change the state of the sending process to PR_SND and set the flag sndflag to 1 which indicates that the process has a valid message to send. And this I put the sending process into the sleepq. It has to sleep in the sleep queue depending upon the value given in the maxwait. And apart from this, I have maintained a per process array[a queue], which holds the processes in queue waiting to send a message to the receiver. This queue I have implemented has an array . The size of the array is NPROC. So I put the process' pid in the queue. This process is in the PR_SND state. Its state changes if there is a time out, at which point I remove this process from the queue and return TIMEOUT error. And the other way the state of the sending process might change is in receive(). In receive(), when the process has read the data, it picks the process from the queue, and if its in the sleep queue qill be removed from the sleep queue and is put in the readylist. So when this process becomes the currect process, it will be able to send the message to the receiver.

For the case where maxwait == 0 , we have to indefinitely. So I just don't do anything with the process which has a maxwait = 0 . I just insert it into the queue of the receiver(i.e the queue for waiting send processes).

These have ripple effect in other parts of the code. I am inserting into the sleepqueue a process which has a state of PR_SND. So the unsleep() function needed to be modified to consider this case. [This is because in unsleep(), when we are trying to remove a process from the sleep queue, it checks the state of the process. Any other state other than the Sleep and RECTIM process will return with an error]. So I have modified unsleep(). And also the kill.c() needs to modified.

Since if a receiver is killed, all the processes that are waiting to send must be freed from the queue.

Following are the test cases I executed :

Test Case1:

I have created a sender and a receiver. The sender sends two messages to the same receiver. Both the messages are printed.

```
void sendb1(pid32 pid1,umsg32 msg1, umsg32 msg2)
{
  sendb(pid1, msg1, 10);
  sendb(pid1, msg2, 15);
}

void recvb1()
{
 umsg32 msg;
 int i = 0;
 for(i=0;i<2;i++)
 {
  msg = receive();
  kprintf("The message is %d\n", msg);
  }
}
  pid32 pid1, pid2;
   pid1 = create(recvb1,1024,20,"r1",0);
```

```
  pid2 = create(sendb1,1024,20,"s1",3,pid1,'a','b');
  resume(pid2);
  resume(pid1);
```

Output:

The message is 97

The message is 98

Test Case 2:

I have created a sender which sends four different messages to the same receiver. I have given the maxwait values as shown below. One of the sendb calls will be timed out and this give a TIMEOUT Error. Except this message all the other messages will be printed.

```
void sendb1(pid32 pid1,umsg32 msg1, umsg32 msg2, umsg32 msg3, umsg32
msg4)
{
sendb(pid1, msg1, 10);
 sendb(pid1, msg2, 15);
  sendb(pid1, msg3, 1);
   sendb(pid1, msg4, 0);
}

void recvb1()
{
 umsg32 msg;
 int i = 0;
```

```
 for(i=0;i<4;i++)
 {
  msg = receive();
  kprintf("The message is %d\n", msg);
  }
}
  pid32 pid1, pid2;
   pid1 = create(recvb1,1024,20,"r1",0);
   pid2 = create(sendb1,1024,20,"s1",5,pid1,'a','b',1,2);
   resume(pid2);
   resume(pid1);
```

Output:

The message is 97

The message is 98

The message is 2

Test Case 3:

I have created 2 receivers. 1 receiver is sent 4 messgaes from four different senders and another receiver is sent one message from one sender. All the messages are delivered to the receivers.

```
void sendb2(pid32 pid1, umsg32 msg1, int32 maxwait)
{
 sendb(pid1, msg1, maxwait);
 }
```

```
void recvb2()
 {
  umsg32 msg;
  msg = receive();
  kprintf("The message is %d\n", msg);
  }



void recvb1()
{
 umsg32 msg;
 int i = 0;
 for(i=0;i<4;i++)
 {
  msg = receive();
  kprintf("The message is %d\n", msg);
  }
}
pid32 snd1, snd2,snd3,snd4,snd5, rcv1, rcv2;
   rcv1 = create(recvb1,1024,20,"r1",0);


   rcv2 = create(recvb2,1024,20,"r1",0);
   snd1 = create(sendb2,1024,20,"s1",3,rcv2,'a', 10);
```

```
snd2 = create(sendb2,1024,20,"s2",3,rcv1,'b',5);
snd3 = create(sendb2,1024,20,"s3",3,rcv1,'c',15);
snd4 = create(sendb2,1024,20,"s4",3,rcv1,'d',0);
snd5 = create(sendb2,1024,20,"s5",3,rcv1,'e',8);

resume(snd2);
resume(snd1);
resume(snd3);
resume(snd4);
resume(rcv1);
resume(snd5);
resume(rcv2);
```

Output:

The message is 98

The message is 97

The message is 99

The message is 100

The message is 101

Test Case 4:

There is only one sender sending one message to one of the receiver. 2 reciver processes are created. One of them gets the message. The other one waits for the message.

```
void sendb2(pid32 pid1, umsg32 msg1, int32 maxwait)
{
 sendb(pid1, msg1, maxwait);
 }

void recvb2()
 {
  umsg32 msg;
  msg = receive();
  kprintf("The message is %d\n", msg);
  }


void recvb1()
{
 umsg32 msg;
 int i = 0;
 for(i=0;i<4;i++)
 {
  msg = receive();
  kprintf("The message is %d\n", msg);
  }
}
pid32 snd1, snd2,snd3,snd4,snd5, rcv1, rcv2;
```

```
  rcv1 = create(recvb2,1024,20,"r1",0);


  rcv2 = create(recvb2,1024,20,"r1",0);
  snd1 = create(sendb2,1024,20,"s1",3,rcv1,'a', 10);


  resume(snd1);
  resume(rcv1);
 resume(rcv2);
```

Output:

The message is 97