typedef struct \_\_eventStruct

{

int eventNum;

wait\_queue\_head\_t \*p;

struct \_\_eventStruct \*next;

} \_\_eventNode;

\_\_eventNode \*lpmyevent\_head = NULL;

\_\_eventNode \*lpmyevent\_end = NULL;

\_\_eventNode \* scheventNum(int eventNum,\_\_eventNode \*\*prev)

{

\_\_eventNode \*tmp = lpmyevent\_head;

\*prev = NULL;

while(tmp)

{

if(tmp->eventNum == eventNum)

return tmp;

\*prev = tmp;

tmp =tmp->next;

}

return NULL;

}

asmlinkage int sys\_eventopen(int eventNum)

{

\_\_eventNode \*newNode;

\_\_eventNode \*prevNode;

if(eventNum)

{

if(!scheventNum(eventNum,&prevNode))

return -1;

else

return eventNum;

}

else

{

newNode = (\_\_eventNode \*)kmalloc(sizeof(\_\_eventNode),GFP\_KERNEL);

newNode->p = (wait\_queue\_head\_t \*)kmalloc(sizeof(wait\_queue\_head\_t),GFP\_KERNEL);

newNode->next = NULL;

newNode->p->task\_list.next = &newNode->p->task\_list;

newNode->p->task\_list.prev = &newNode->p->task\_list;

if(!lpmyevent\_head)

{

newNode->eventNum =2;

lpmyevent\_head =lpmyevent\_end =newNode;

return newNode->eventNum;

}

else

{

newNode->eventNum = lpmyevent\_end->eventNum + 2;

lpmyevent\_end->next = newNode;

lpmyevent\_end = newNode;

}

return newNode->eventNum;

}

return 0;

}

asmlinkage int sys\_eventsig(int eventNum)

{

\_\_eventNode \*tmp = NULL;

\_\_eventNode \*prev = NULL;

if(!(tmp = scheventNum(eventNum,&prev)))

return;

wake\_up(tmp->p);

return 1;

}

asmlinkage int sys\_eventwait(int eventNum)

{

\_\_eventNode \*tmp;

\_\_eventNode \*prev = NULL;

wait\_queue\_t wait;

unsigned long flags;

if(tmp = scheventNum(eventNum,&prev))

{

wait.tasks = current;

current->state = TASK\_UNINTERRUPTIBLE;

write\_lock\_irqsave(&tmp->p->lock,flags);

\_\_add\_wait\_queue(tmp->p,&wait);

write\_unlock(&tmp->p-lock);

schedule();

write\_lock\_irq(&tmp->p->lock);

\_\_remove\_wait\_queue(tmp->p,&wait);

write\_unlock\_irqrestore(&tmp->p->lock,flags);

}

}

asmlinkage int sys\_eventclose(int eventNum)

{

\_\_eventNode \*prev;

\_\_eventNode \*releaseItem;

sys\_eventsig(eventNum);

if(releaseItem = scheventNum(eventNum,&prev))

{

if (releaseItem == lpmyevent\_end)

lpmyevent\_end = prev;

if (releaseItem == lpmyevent\_head)

{

lpmyevent\_head = lpmyevent\_head->next;

goto wake;

}

prev->next = releaseItem->next;

}

wake:

if(releaseItem)

{

kfree(releaseItem);

return 1;

}

return 0;

}