8 Communication In Operating Systems

- Inter-process communication: important mechanism in operating systems
- Unix-Example : shell-pipeline : IPC between user processes
- several mechanisms
  - Semaphores for coordination/synchronization ([Com83], chapter 6)
  - Terminals [Hoa74][Han 75]:
    * Mechanism in mutual exclusion
    * high-level, programming language constuctions, some kind of abstract data type or object
    * no system call
  - Data transmission by using shared variables
  - message passing
    * synchronization and data transmission

8.1 Message Passing

- Form of inter-process Communication /process synchronization /-coordination
- alternative : shared variables
- unlike synchronization by using semaphores : it can be asynchronous
- implementation using system calls : *sendreceive*
- several variants is possible
  - direct message passing
  - message sending and receiving can be blocking or unblocking
  - rendezvous: send and receive are blocking
  - Capacity of the binding (buffer size): what would happen, if buffer is full ?
  - Messages of determined or variable size
  - more than one receiver is possible ?
  - specified sending/receiving process
  - ....

8.2 Message Passing In Xinu

- Tow forms of message passing
  1. process-to-process (direct)
  2. message left at redezvous points (chapter 14, [Com83])
- Three system calls
  1. send: (asynchron)
  2. receive (synchron)
  3. recvclr (asynchron)
- receiving Buffer of size 1 (= one word),
  1. only the first is received
  2. all other are lust because sender dose not blocks
- new process state: receiving (**PRREC**)
- storage place: in the process table entry.
  - not in the sender memory because sending process might exit before message is received
  - not in the recipient's memory because it poses a security threat

```
/* proc.h   see P. 55 */
....

struct  pentry {
        char     pstate;   /* process state                    */
        ...
        short    pmsg;     /* 1 message sent to this process */
        short    phasmsg; /* nonzero => msg is valid         */
        ...
};
...
```

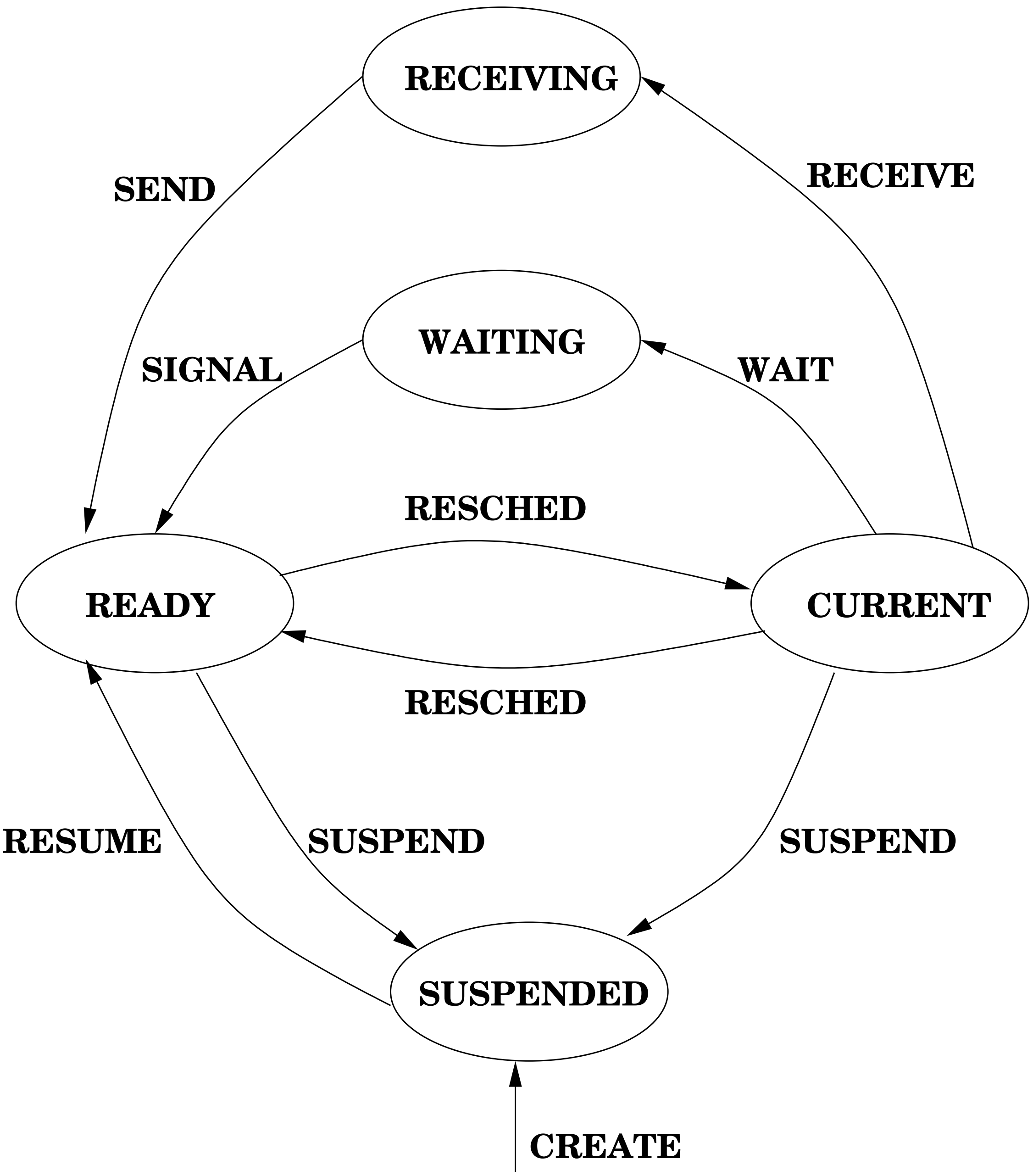

Figure 7.1: Process state transitions for the receiving state

8.3 Implementation Of Send

- Interrupt disabled
- Uses the process id to access corresponding process table entry
- Errors when
  - invalid process id
  - receiving buffer is full
- else
  1. passing the message
  2. setting the flag (phasmsg)
  3. if the recipient is waiting for a message, the reschedule by calling **ready()**

Sending – send

```
SYSCALL send(pid, msg)
   ....   {  ....
      struct   pentry  *pptr;/* receiver's proc. table addr.*/

       disable(ps);
       if (isbadpid(pid)
           || ( (pptr= &proctab[pid])->pstate == PRFREE)
           || pptr->phasmsg) {
                restore(ps);
                return(SYSERR);
        }
        pptr->pmsg = msg;       /* deposit message            */
        pptr->phasmsg = TRUE;
                              /* if receiver waits, start it */
        if (pptr->pstate == PRRECV)
                ready(pid, RESCHYES);
        restore(ps);
        return(OK);
}
```

8.4 Implementation Of Receive

Asynchronous receiving – recvclr

- like the synchronous receiving
- if process has message: return
- else return(OK)

```
SYSCALL recvclr()
{
        char    ps;
        int     msg;

        disable(ps);
        if ( proctab[currpid].phasmsg ) {/* existing message? */
                proctab[currpid].phasmsg = FALSE;
                msg = proctab[currpid].pmsg;
        } else
                msg = OK;
        restore(ps);
        return(msg);
}
```

Synchronous receiving – receive

- Like the asynchronous receiving
- deference: if process has no message: changes P to the receiving state and calls receiving state **resched**.

HIER KOMMT EINE FIGURE (COM P. 95)

Synchronous receiving

```
SYSCALL receive()
{
        struct  pentry  *pptr;
        int     msg;
        char    ps;

        disable(ps);
        pptr = &proctab[currpid];
        if ( !pptr->phasmsg ) { /* if no message, wait for one */
                pptr->pstate = PRRECV;
                resched();
        }
        msg = pptr->pmsg;      /* retrieve message            */
        pptr->phasmsg = FALSE;
        restore(ps);
        return(msg);
}
```