Embedded Operating Systems

Trainer: Nilesh Ghule

Message Queue

p1.c → p1.out

```
msg_op
ty - 11
n1 - v
n2 - v
```

```
id = msgget( key, IPC_CREAT|0600);
msgsnd(id, &mop, sz, 0);
msgrcv(id, &mres, sz, 22, 0);
msgctl(id, IPC_RMID, NULL);
```

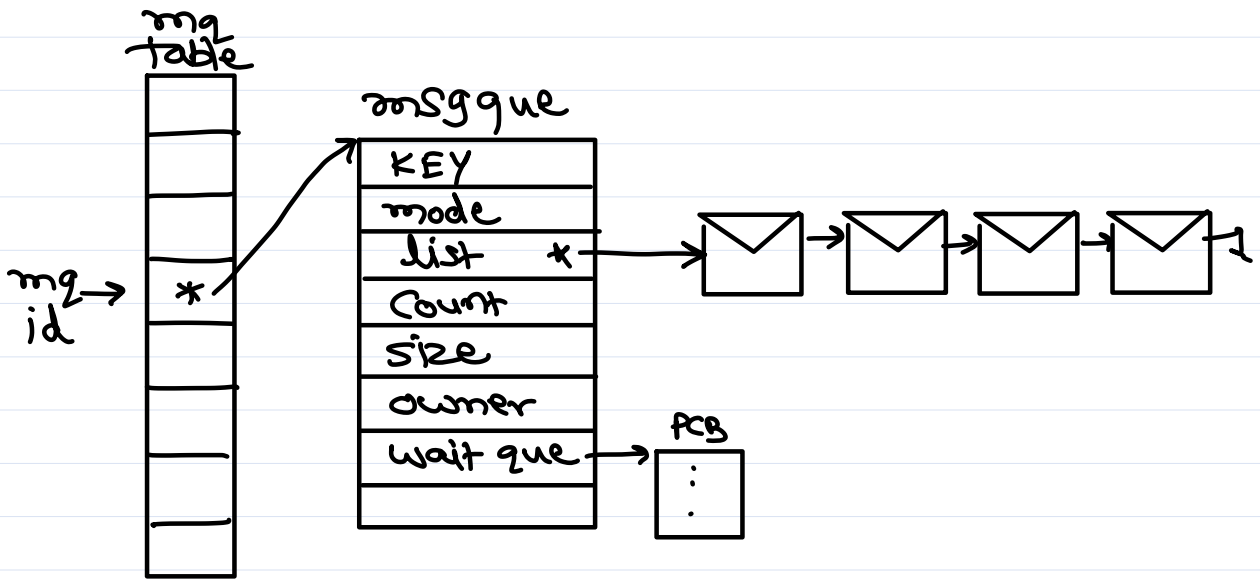
p.h

1. key
2. mop
3. mres
4. mtypes

p2.c → p2.out

```
msg_res
ty - 22
res - v
```

```
id = msgget(key, 0);
msgrcv(id, &mop, sz, 11, 0);
...
msgsnd(id, &mres, sz, 0);
```



① mqid = msgget(key, flags);
- @ create msgque obj in kernel space to keep info about mq & init it.
- ⓑ store its addr in mq table.
- © return index of mq table's slot - mqid

flags - 0: get mqid of mq already created.
IPC_CREAT|mode: to create new mq.

② msgctl(mqid, IPC_RMID, NULL);
- @ release mem of all msgs in list.
- ⓑ wake up all process sleeping on msgque.
- © release msgque struct & remove entry from mq table.

③ msgctl(mqid, IPC_STAT, &info); struct (out) msqid_ds
- @ fill info from mq obj into the given out param(arg3).

```
cmd> ipcs -?
cmd> ipcrm -?
```

Message Queue

④ msgsnd(mqid, &msg, msg_sz, flags);
- ⓐ allocate a msg struct in kernel space and copy data from user space struct (given in arg2).
- ⓑ add that msg into msg list
- ⓒ update info in msg que obj
- ⓓ wakeup any process that is waiting for the given msg type.

⑤ msgrcv(mqid, &msg, msg_sz, type, flags);
- ⓐ check if msg of given type (arg4) is available in msg list.
- ⓑ if available, copy it into user space msg struct out param (arg2).
- ⓒ if not available, make current process sleep in mq's wait que. after wakeup get the msg & then return.

p1.c → p1.out

```
msg-op
ty - 11
m1 - v
m2 - v
```

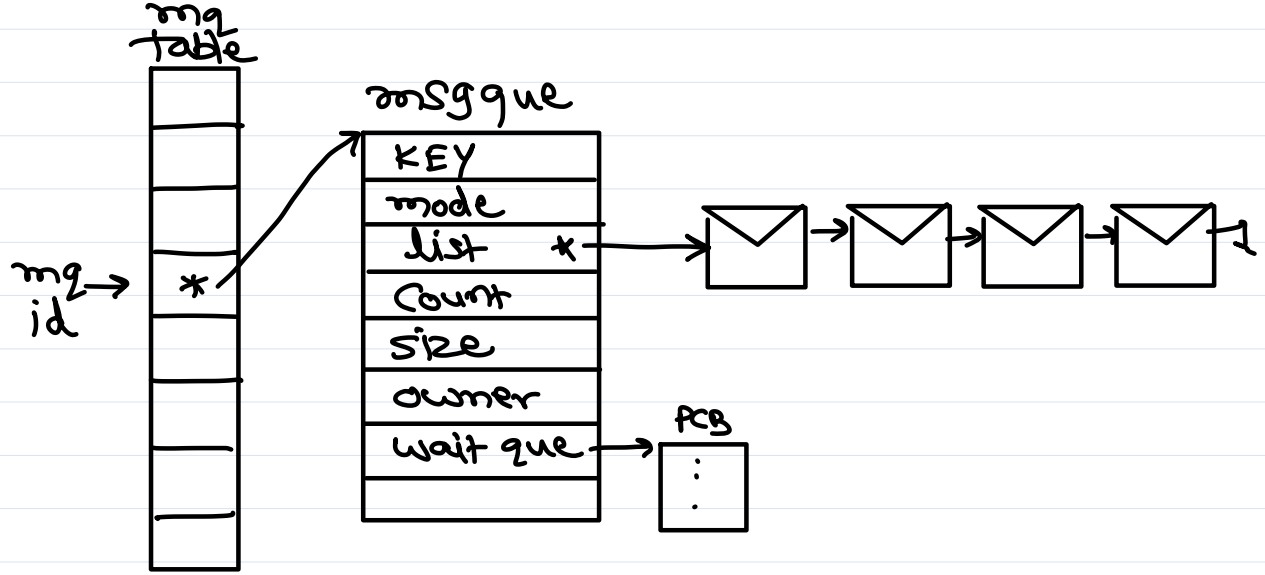
p.h
① key
② mop
③ mres
④ m types

p2.c → p2.out

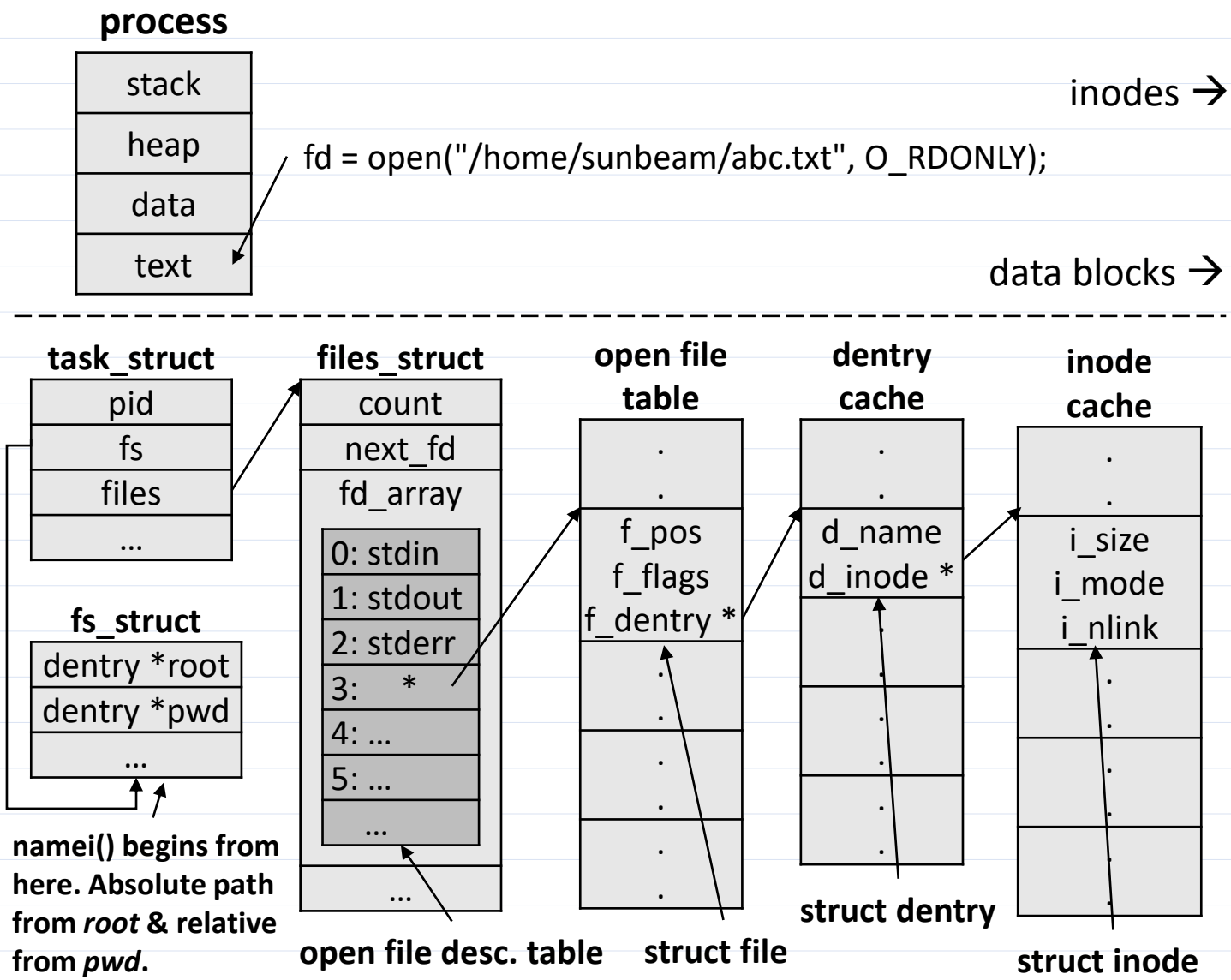
```
msg_res
ty - 22
res - v
```

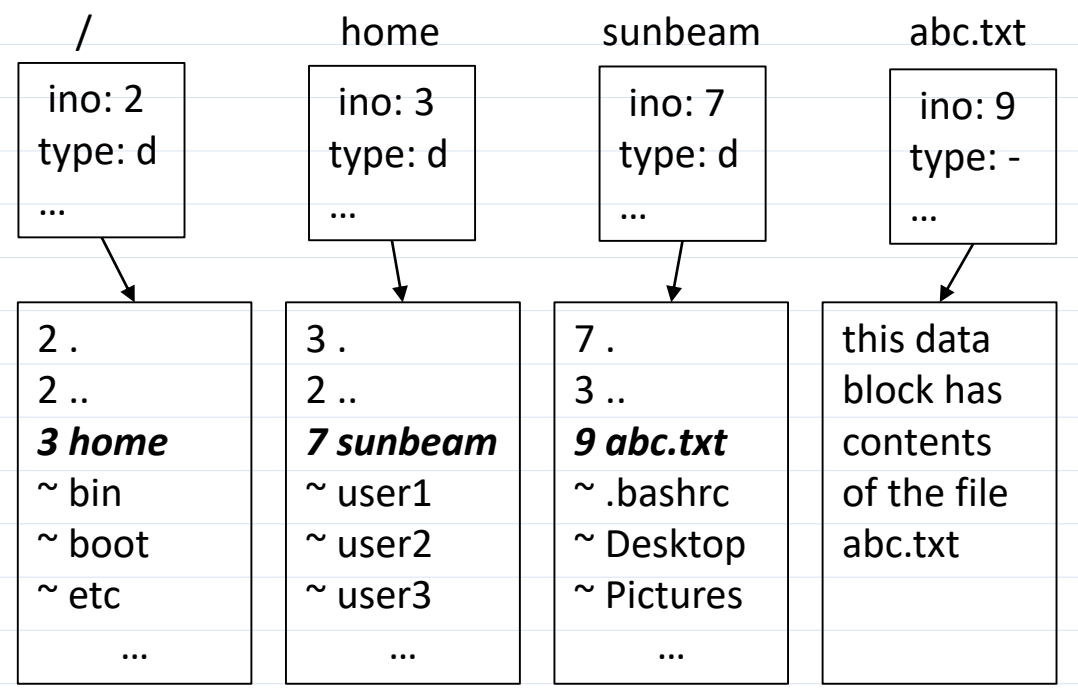
```
id=msgget( key, IPC_CREAT|0600);
msgsnd(id, &mop, sz, 0);
msgrcv(id, &mres, sz, 22, 0);
msgctl(id, IPC_RMID, NULL);
```

```
id=msgget(key, 0);
msgrcv(id, &mop, sz, 11, 0);
...
msgsnd(id, &mres, sz, 0);
```



Linux - open() syscall internals (simplified)



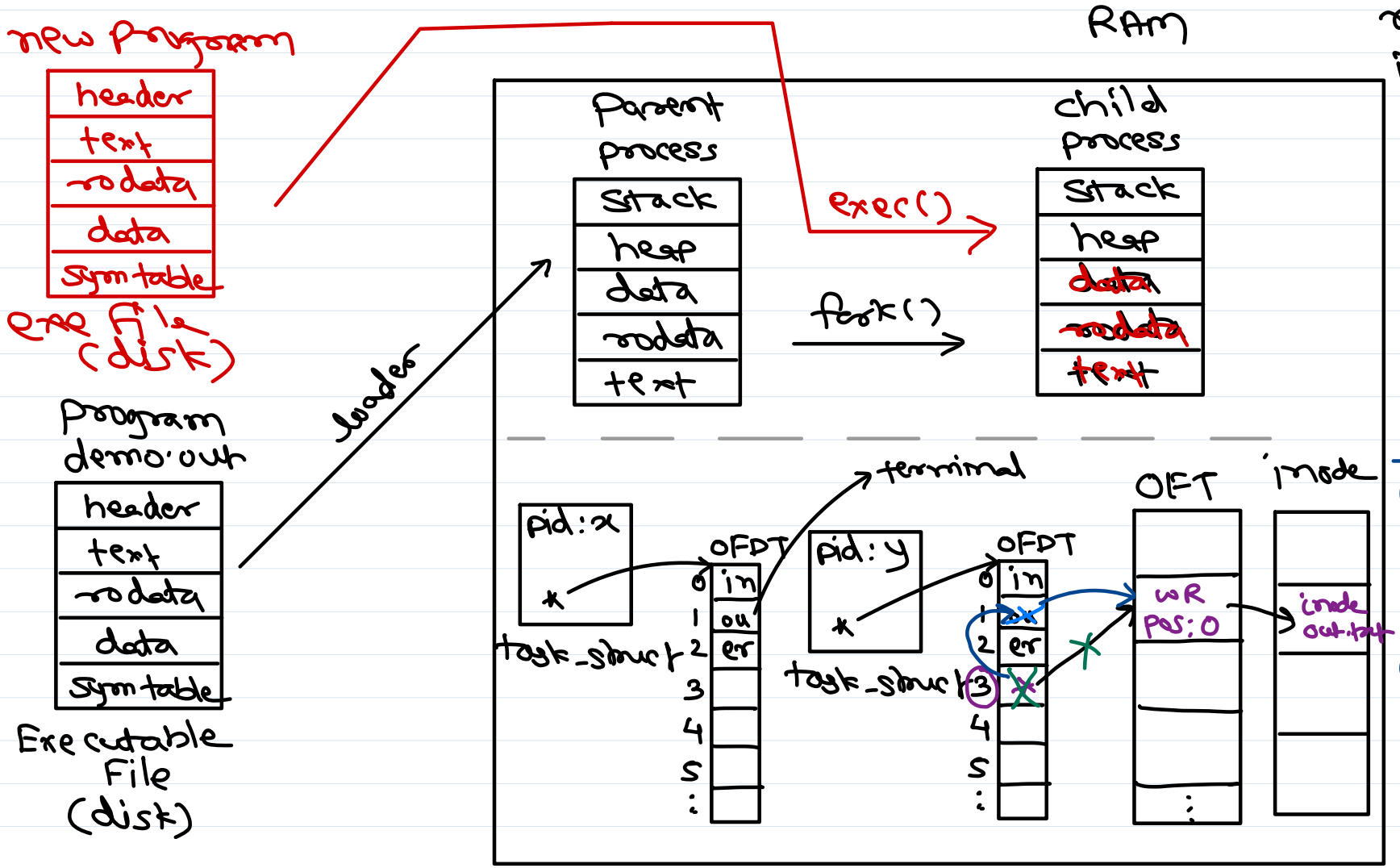


open() syscall internal steps (in short):

1.convert file path into inode number: namei() algo.
2.load inode from disk inode list into inode cache.
3.create open file entry into open file table.
4.make entry into process's open file desc table.
5.return index of file desc table as file descriptor.

This returned file descriptor is used for further calls to read(), write(), lseek(), close(), etc.

IO Redirection

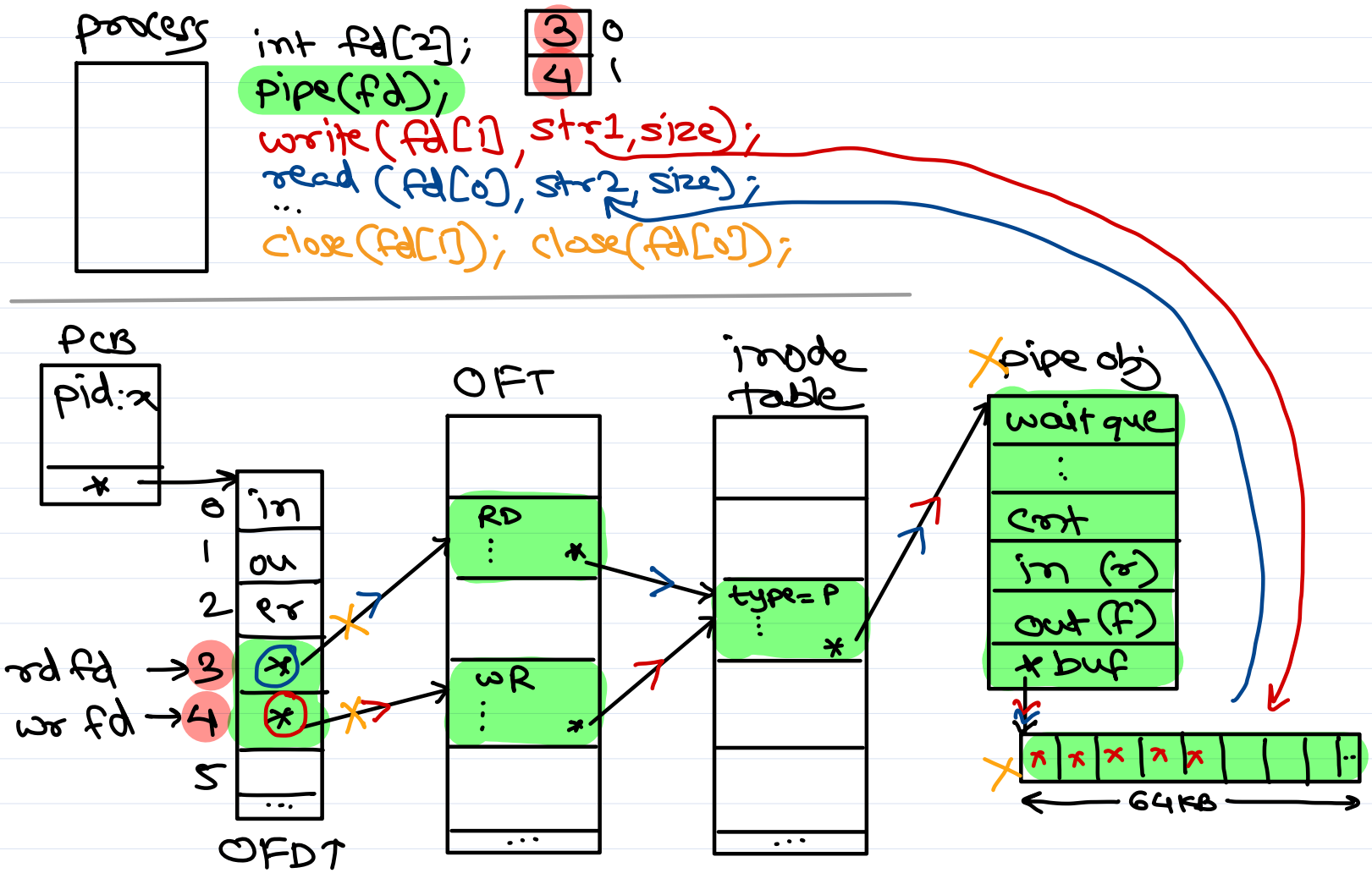



```
ret = fork();
if (ret == 0) {
    ③ fd = open("out.txt", O_WR);
      close(1); //stdout
      dup(fd);
      close(fd);
    err = exec("child", ...);
    //...
}
```

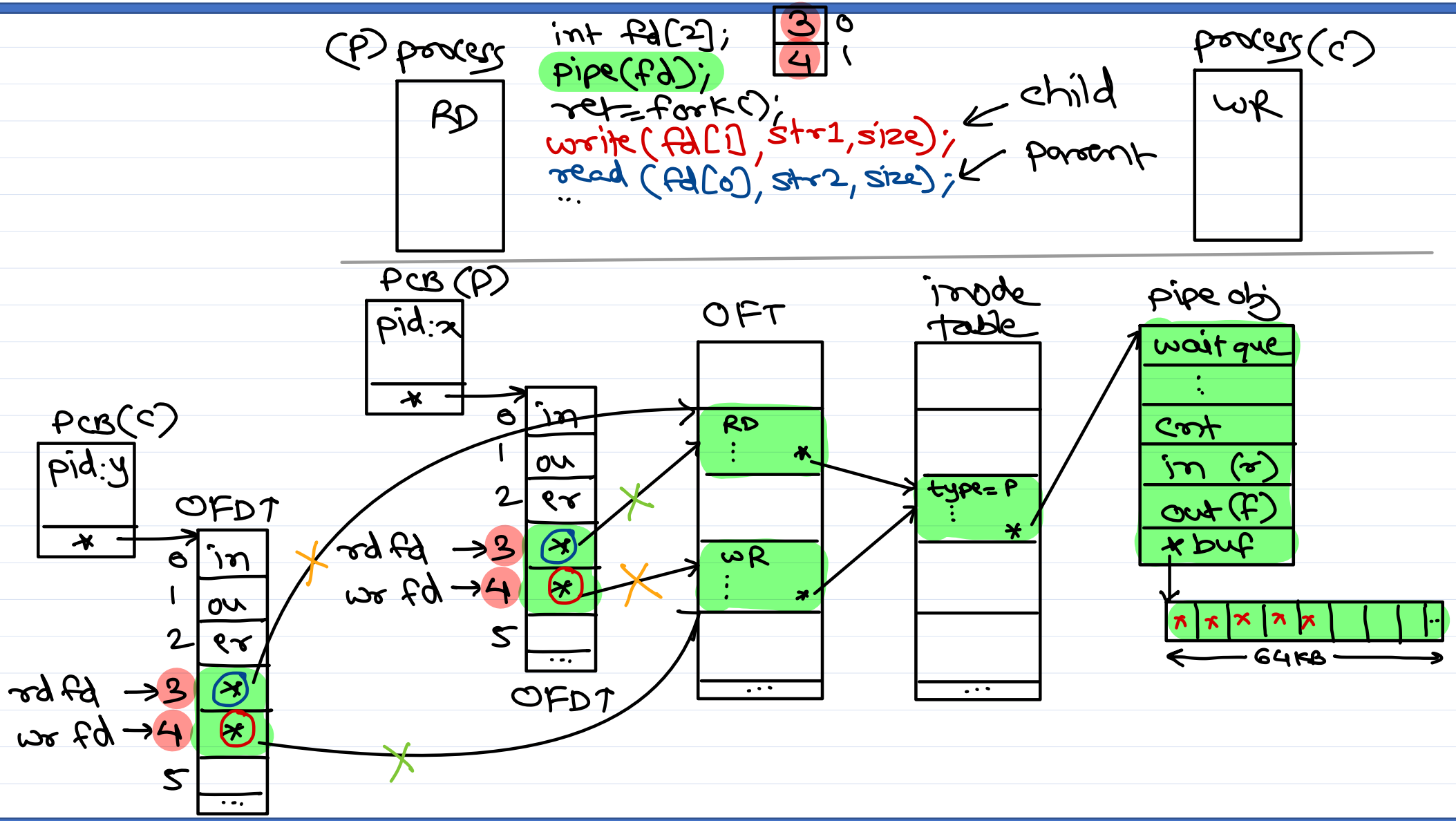
① dup(fd): copies given fd on lowest numbered avail desc in OFDT of current process

② dup2(fd, new_fd): copies given fd on given new_fd. If new_fd is in use, it will be first closed.
e.g. dup2(fd, 1);

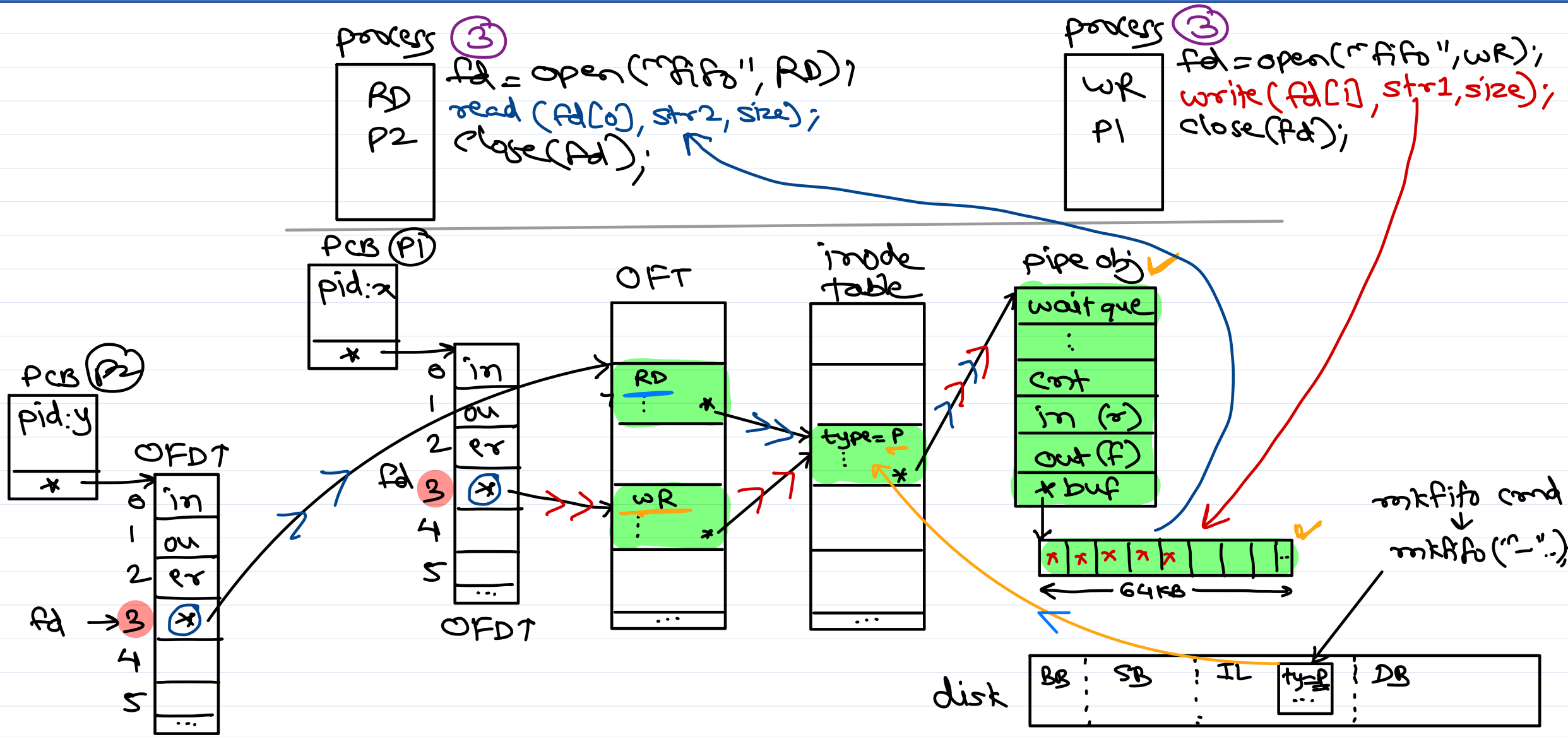
Pipe


process
int fd[2];
pipe(fd);
write(fd[1], str1, size);
read(fd[0], str2, size);
close(fd[1]); close(fd[0]);
3
4
0
1
PCB
pid:x
OFT
inode table
pipe obj
wait que
cnt
in (r)
out (f)
*buf
RD
WR
type=P
in
ou
pr
rd fd
wr fd
OFDT
64KB

Pipe - IPC


(P) process
RD
int fd[2];
pipe(fd);
ret=fork();
write(fd[1], str1, size);
read(fd[0], str2, size);
...
child
parent
process (C)
WR
PCB (P)
pid:x
PCB(C)
pid:y
OFDT
in
ou
er
rd fd → 3
wr fd → 4
OFT
RD
WR
inode table
type= P
pipe obj
wait que
cnt
in (r)
out (f)
*buf
64KB

FIFO - IPC


process 3
RD
P2
fd = open("fifo", RD);
read (fd[0], str2, size);
close(fd);
process 3
WR
P1
fd = open("fifo", WR);
write(fd[1], str1, size);
close(fd);
PCB P1
pid:x
PCB P2
pid:y
OFDT
in
ou
er
OFT
RD
WR
inode table
type = P
pipe obj
wait que
cnt
in (r)
out (f)
*buf
64KB
mkfifo cmd
mkfifo("fifo",...);
disk
BB
SB
IL
DB

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>