CS 1217

Lecture 4 – Process Creation, Termination

Logistics

- Assignment # 2 goes out this week

- Mandatory lab hours, starting this week
  - Tuesday 6:30 – 8 pm
  - Will send out the room details shortly

Recap: pmap

- Information about memory mappings of process

```
cs304@cs304-devel:~$ pmap 2002
2002:   bash
0000560445ea5000   1040K r-x-- bash
00005604461a8000     16K r---- bash
00005604461ac000     36K rw--- bash
00005604461b5000     40K rw---   [ anon ]
00005604480ff000   1604K rw---   [ anon ]
00007fb4b8a15000   1948K r-x-- libc-2.27.so
00007fb4b8bfc000   2048K ----- libc-2.27.so
00007fb4b8dfc000     16K r---- libc-2.27.so
00007fb4b8e00000      8K rw--- libc-2.27.so
00007fb4b945d000      4K rw---   [ anon ]
00007fff083df000    132K rw---   [ stack ]
```

Aside /proc

```
root@cs304-devel:/proc# ls
1     1097  1163  1407  1585  1648  20    269   297  525  694   asound       kallsyms      schedstat
10    11    1194  1415  1586  165   21    27    298  537  7     buddyinfo    kcore         scsi
1000  1100  12    1420  1588  1685  2151  270   299  543  766   bus          keys          self
1001  1102  1219  1431  1589  1686  219   271   30   544  78    cgroups      key-users     slabinfo
1012  1104  1222  1480  1594  1690  22    272   308  549  780   cmdline      kmsg          softirqs
1014  1107  1223  1484  1596  17    2222  2764  309  562  788   consoles     kpagecgroup   stat
1016  1109  1236  1486  16    1724  2232  2779  31   568  79    cpuinfo      kpagecount    swaps
1022  1110  1240  1488  1603  1740  2278  2781  312  569  8     crypto       kpageflags    sys
1029  1111  1242  149   1605  1760  23    2782  314  570  80    devices      loadavg       sysrq-trigger
1038  1112  1247  15    1606  1775  2384  2783  318  579  81    diskstats    locks         sysvipc
1042  1115  1250  1503  1614  1789  24    2794  32   581  82    dma          mdstat        thread-self
1047  1117  13    1507  1615  1798  2409  28    334  583  83    driver       meminfo       timer_list
1049  1121  1358  1518  1616  18    242   2814  336  584  84    execdomains  misc          tty
1052  1122  1359  1548  1618  1839  2495  282   34   585  85    fb           modules       uptime
1054  1123  1363  1554  1622  187   25    285   348  587  89    filesystems  mounts        version
1070  1126  1378  1557  1623  188   26    286   35   6    9     fs           mtrr          version_signature
1073  1131  1381  1566  1625  19    263   289   36   621  98    interrupts   net           vmallocinfo
1075  1134  1386  1571  1627  1911  264   29    4    624  982   iomem        pagetypeinfo  vmstat
1078  1138  1389  1575  163   1984  267   291   411  692  991   ioports      partitions    zoneinfo
1093  115   14    1579  164   2     2671  295   5    693  acpi  irq          sched_debug
```

- Can you read from it?
- Can you write to it?

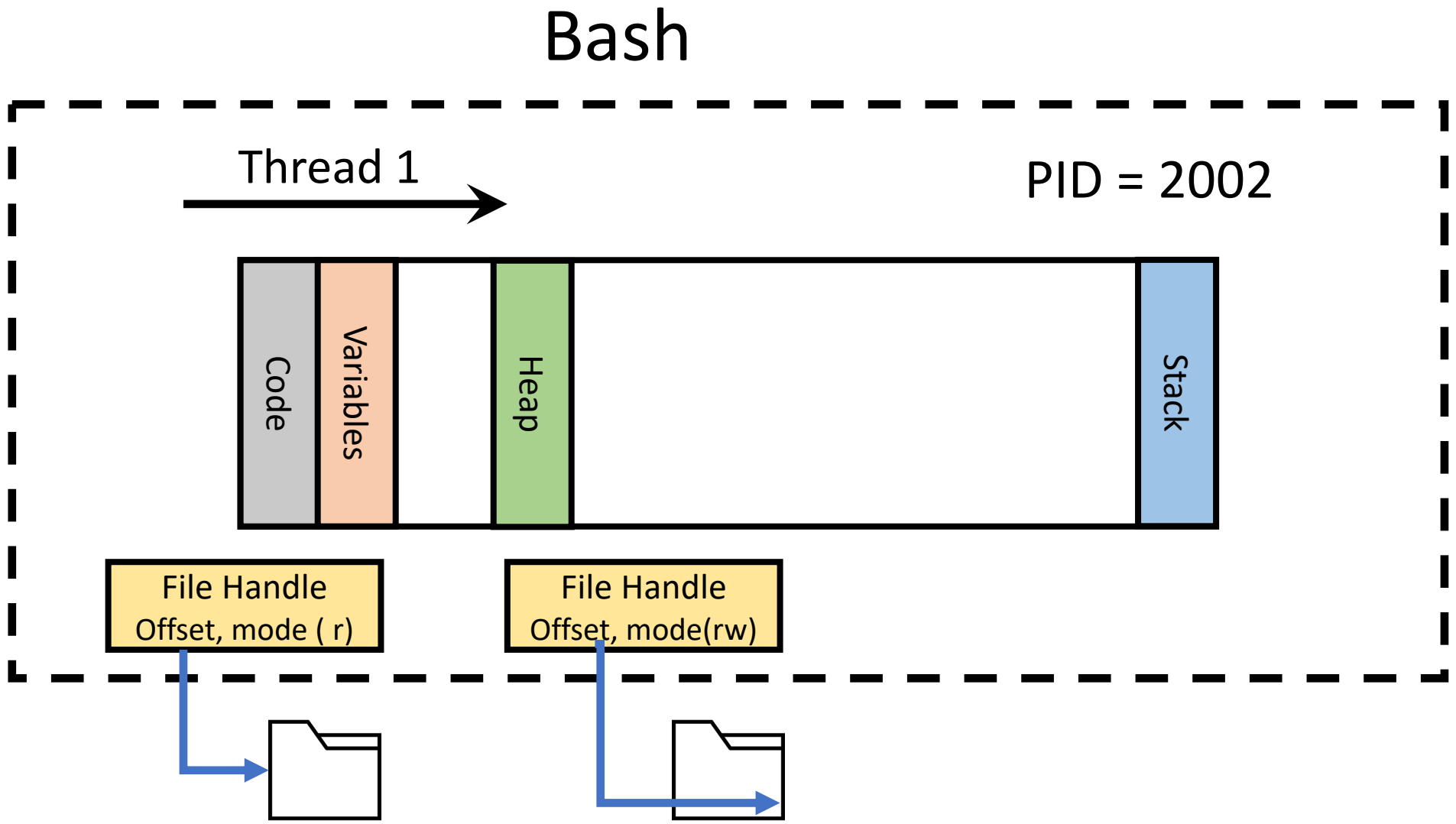
- `echo 3 > /proc/sys/vm/drop_caches`
- (alright, I lied before)

cpuinfo and meminfo

```
cs304@cs304-devel:~$ cat /proc/cpuinfo | more
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 61
model name      : Intel(R) Core(TM) i5-5257U CPU @ 2.70GHz
stepping        : 4
cpu MHz         : 2699.998
cache size      : 3072 KB
physical id     : 0
siblings        : 1
core id         : 0
cpu cores       : 1
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 20
wp              : yes
```

```
cs304@cs304-devel:~$ cat /proc/meminfo
MemTotal:        2041304 kB
MemFree:          193196 kB
MemAvailable:     925720 kB
Buffers:           54232 kB
Cached:           790040 kB
SwapCached:            0 kB
Active:          1335996 kB
Inactive:         354308 kB
Active(anon):     846872 kB
Inactive(anon):     5368 kB
Active(file):     489124 kB
Inactive(file):   348940 kB
Unevictable:          16 kB
Mlocked:              16 kB
SwapTotal:        728520 kB
SwapFree:         728520 kB
Dirty:                92 kB
Writeback:             0 kB
AnonPages:        846056 kB
Mapped:           225512 kB
Shmem:              6212 kB
Slab:              83600 kB
SReclaimable:      52028 kB
SUnreclaim:        31572 kB
KernelStack:        7820 kB
PageTables:        38140 kB
NFS_Unstable:          0 kB
```

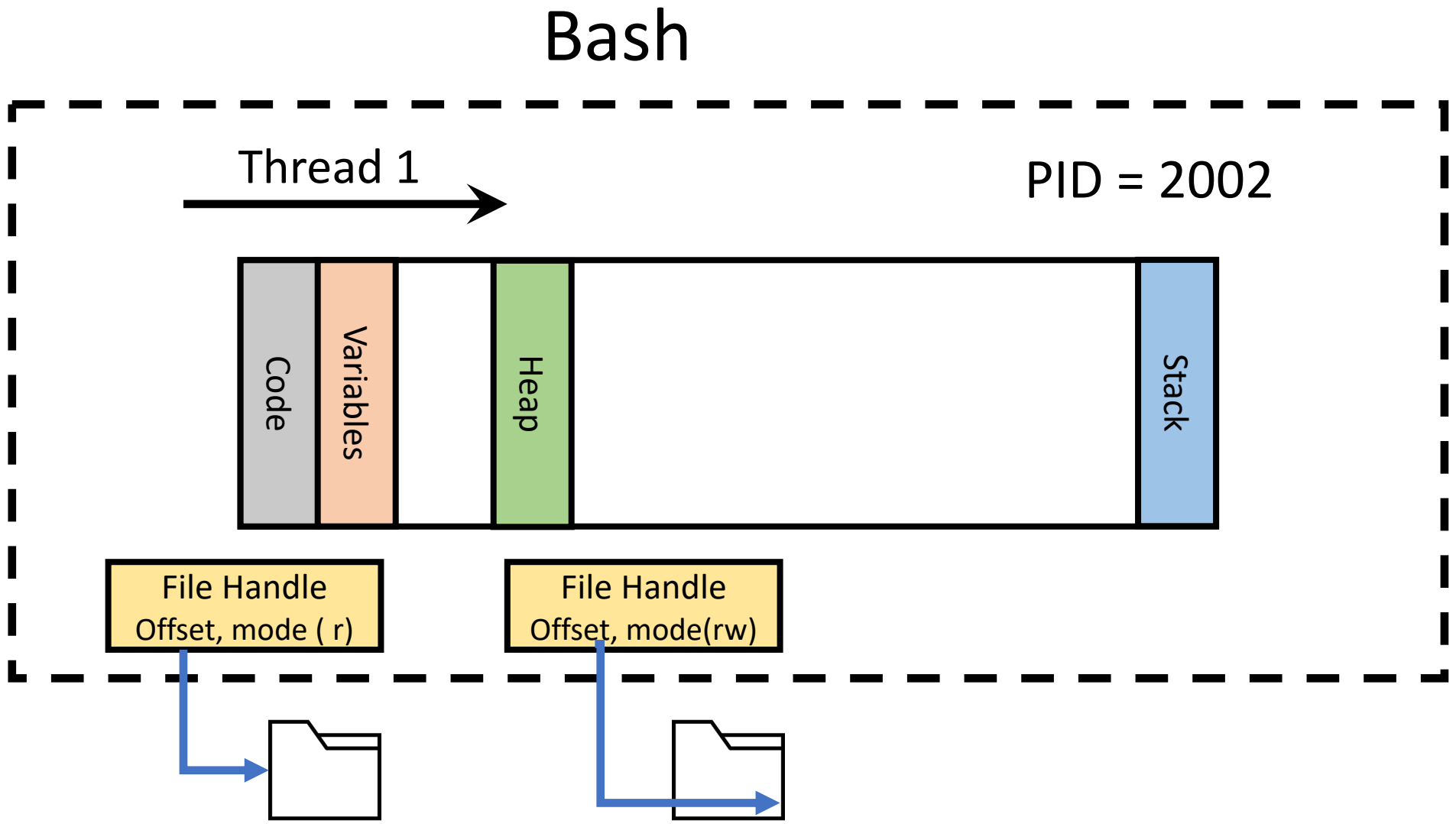
The Process Abstraction



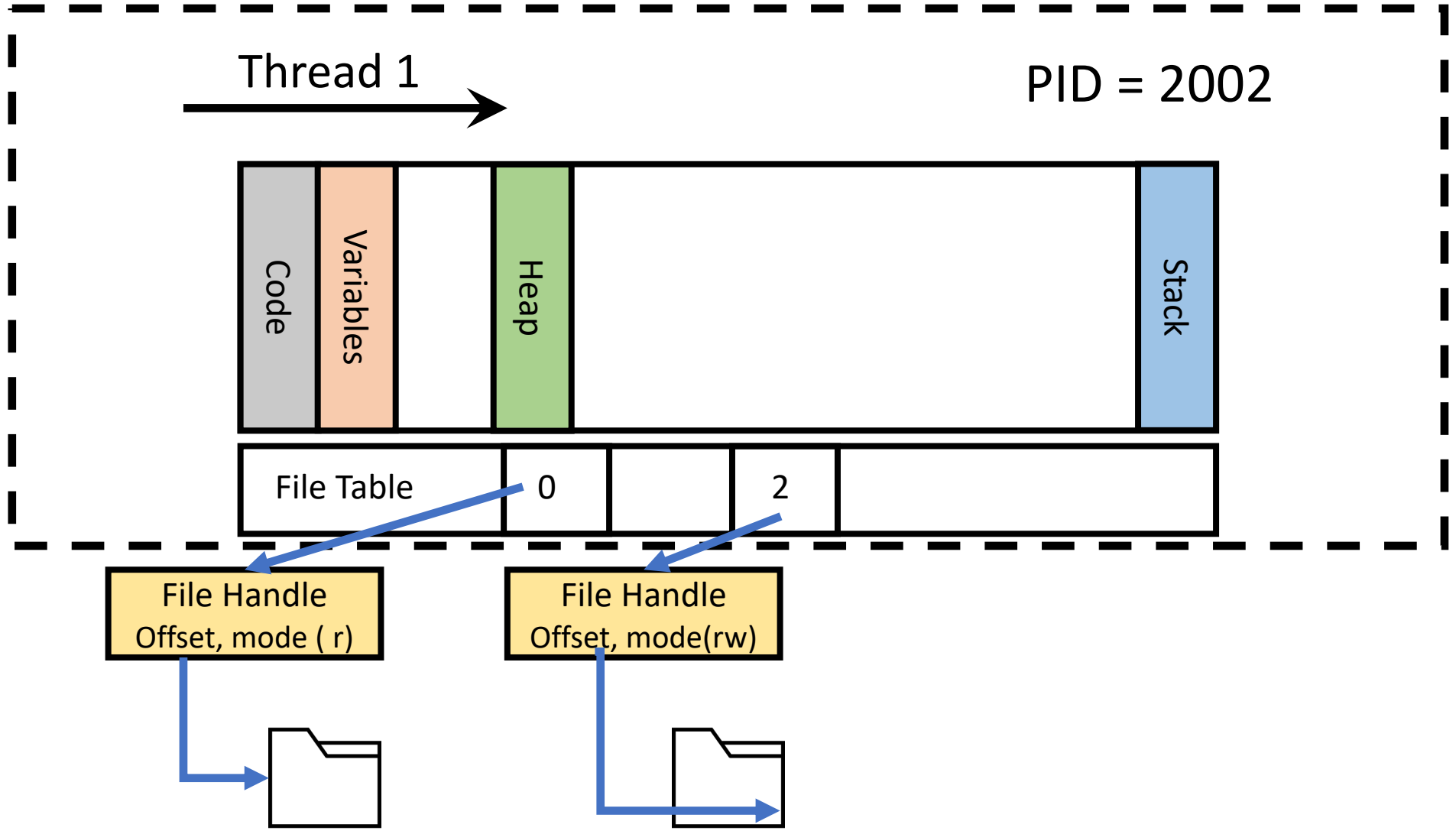
Aside: lsof example

```
cs304@cs304-devel:~$ lsof -p 2002 | more
COMMAND  PID  USER   FD   TYPE DEVICE SIZE/OFF   NODE NAME
bash    2002 cs304  cwd    DIR    8,1     4096 426011 /home/cs304
bash    2002 cs304  rtd    DIR    8,1     4096      2 /
bash    2002 cs304  txt    REG    8,1  1113504 262153 /bin/bash
bash    2002 cs304  mem    REG    8,1    47568 268654 /lib/x86_64-linux-gnu/libnss_files-2.27.so
bash    2002 cs304  mem    REG    8,1    97176 268648 /lib/x86_64-linux-gnu/libnsl-2.27.so
bash    2002 cs304  mem    REG    8,1    47576 268665 /lib/x86_64-linux-gnu/libnss_nis-2.27.so
bash    2002 cs304  mem    REG    8,1    39744 268650 /lib/x86_64-linux-gnu/libnss_compat-2.27.so
bash    2002 cs304  mem    REG    8,1 10281936 794322 /usr/lib/locale/locale-archive
bash    2002 cs304  mem    REG    8,1  2030544 268564 /lib/x86_64-linux-gnu/libc-2.27.so
bash    2002 cs304  mem    REG    8,1    14560 268587 /lib/x86_64-linux-gnu/libdl-2.27.so
bash    2002 cs304  mem    REG    8,1   170784 268722 /lib/x86_64-linux-gnu/libtinfo.so.5.9
bash    2002 cs304  mem    REG    8,1   170960 268536 /lib/x86_64-linux-gnu/ld-2.27.so
bash    2002 cs304  mem    REG    8,1    26376 139948 /usr/lib/x86_64-linux-gnu/gconv/gconv-modules
.cache
bash    2002 cs304    0u   CHR  136,1      0t0      4 /dev/pts/1
bash    2002 cs304    1u   CHR  136,1      0t0      4 /dev/pts/1
bash    2002 cs304    2u   CHR  136,1      0t0      4 /dev/pts/1
bash    2002 cs304  255u   CHR  136,1      0t0      4 /dev/pts/1
```

File Handles



File Tables and File Handles



A Bit about File Tables

- file descriptor → file handle.
- file handle → file object.
- file object → blocks on disk

- What are these ”links” called?

- Are these indirections useful? Why?

Why do this?

- The additional level of indirection allows certain pieces of state to be shared separately.
- **File descriptors** are private to each process.
- **File handles** are private to each process but shared after process creation.
  - **File handles** store the current file **offset**, or the position in the file that the next read will come from or write will go to.
  - File handles can be **deliberately** shared between two processes.
- **File objects** hold some file state and may be **shared transparently** between many processes

Where do processes come from?

- The first process is “init”
  - The process that is created at bootup, every other process descends from it
- The OS provides support for existing processes to “create” new processes
  - Parent-child relationships between processes
  - Hence, process trees, check **pstree**
- New process would need the same abstractions as parent
  - CPU state (Threads)
  - Address Spaces (Memory)
  - Open file handles, if need be

The Lifecycle of a Process

- Birth (Create)
  - Who creates a process? What does it mean to create a process?
  - OS Interfaces for process creation
- Death (Destroy)
  - Who destroys a process? What does it mean to destroy a process?
  - What are the OS interfaces to do so
- Wait()
  - Wait for something to happen or complete
- Other Interfaces to a Process
  - Mechanisms to suspend or resume process execution
  - Status query – current state, runtime etc.

Process States

```
27  struct context {
28    uint edi;
29    uint esi;
30    uint ebx;
31    uint ebp;
32    uint eip;
33  };
```

I want to capture ***thread*** state – what do these things represent?

```
35  enum procstate { UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };
36
37  // Per-process state
38  struct proc {
39    uint sz;                     // Size of process memory (bytes)
40    pde_t* pgdir;                // Page table
41    char *kstack;                // Bottom of kernel stack for this process
42    enum procstate state;        // Process state
43    int pid;                     // Process ID
44    struct proc *parent;         // Parent process
45    struct trapframe *tf;        // Trap frame for current syscall
46    struct context *context;     // swtch() here to run process
47    void *chan;                  // If non-zero, sleeping on chan
48    int killed;                  // If non-zero, have been killed
49    struct file *ofile[NOFILE];  // Open files
50    struct inode *cwd;           // Current directory
51    char name[16];               // Process name (debugging)
52  };
```

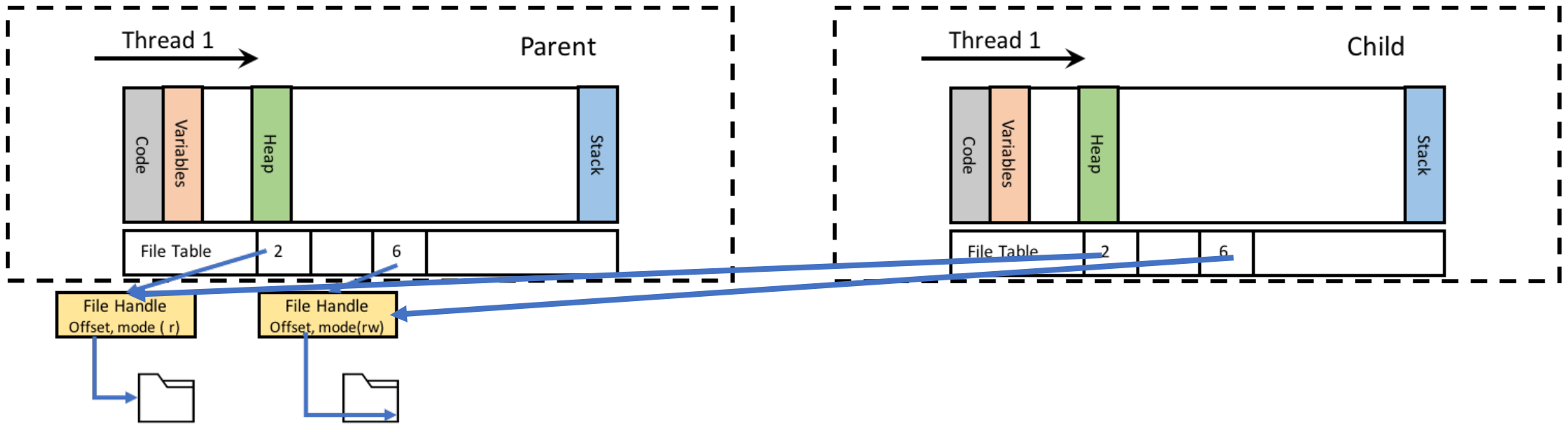
Process Birth: fork()

- fork () allows a process to create a new process

- What does that mean?

- fork() copies the caller process.
- fork() copies the address space.
- fork() copies the process file table.

Process Creation : fork()



Process creation : fork()

```
int returnCode = fork();

if (returnCode == 0) {
    // child (new process)
    printf("hello, I am child (pid:%d)\n", (int) getpid());

} else {
    // parent goes down this path (original process)
    printf("hello, I am parent of %d (pid:%d)\n",
           __________ , (int) getpid());
}
```

- How many times do you think fork() returns?
- What are the return values?