

# Virtual Memory Paging

- An important task of a virtual-memory system is to relocate pages from physical memory out to disk
- Early UNIX systems **swapped out the entire process** at once
- Modern UNIX systems rely more on paging
- In order to do that Linux too use the paging mechanism
- The paging system can be divided to the
  - policy algorithm
  - paging mechanism
- Linux's pageout policy is LFU (least frequently used)

# Virtual Memory Swapping

- The paging mechanism support both paging to dedicated swap devices and to files
- Blocks are allocated from the swap device according to bitmap of used blocks
- The allocator uses the next fit algorithm
- When a page is swapped out the page-not-present bit is set (or the present is unset)
- **cat /proc/swaps**  
shows information about the used swap devices

# kswapd (*file mm/vmscan.c*)

- **kswapd** is a kernel process which reclaim memory from the VM subsystem when memory gets low.
- We need this to make sure that there will always be free memory available for interrupts handlers
- A regular process goes to sleep until the kernel find free memory (from other processes for example)
- We don't want an interrupt handler to sleep, so we keep a certain level of free memory frames.
- If we go below this level the **kswapd** free more memory

# Page buffering (In general)

- Evicted pages are kept on two lists:
  - free and modified page lists
- Pages are read into the frames on the free page list
- Pages are written to disk in large chunks from the modified page list
- If an evicted page is referenced, and it is still on one of the lists, it is made valid at a very low cost

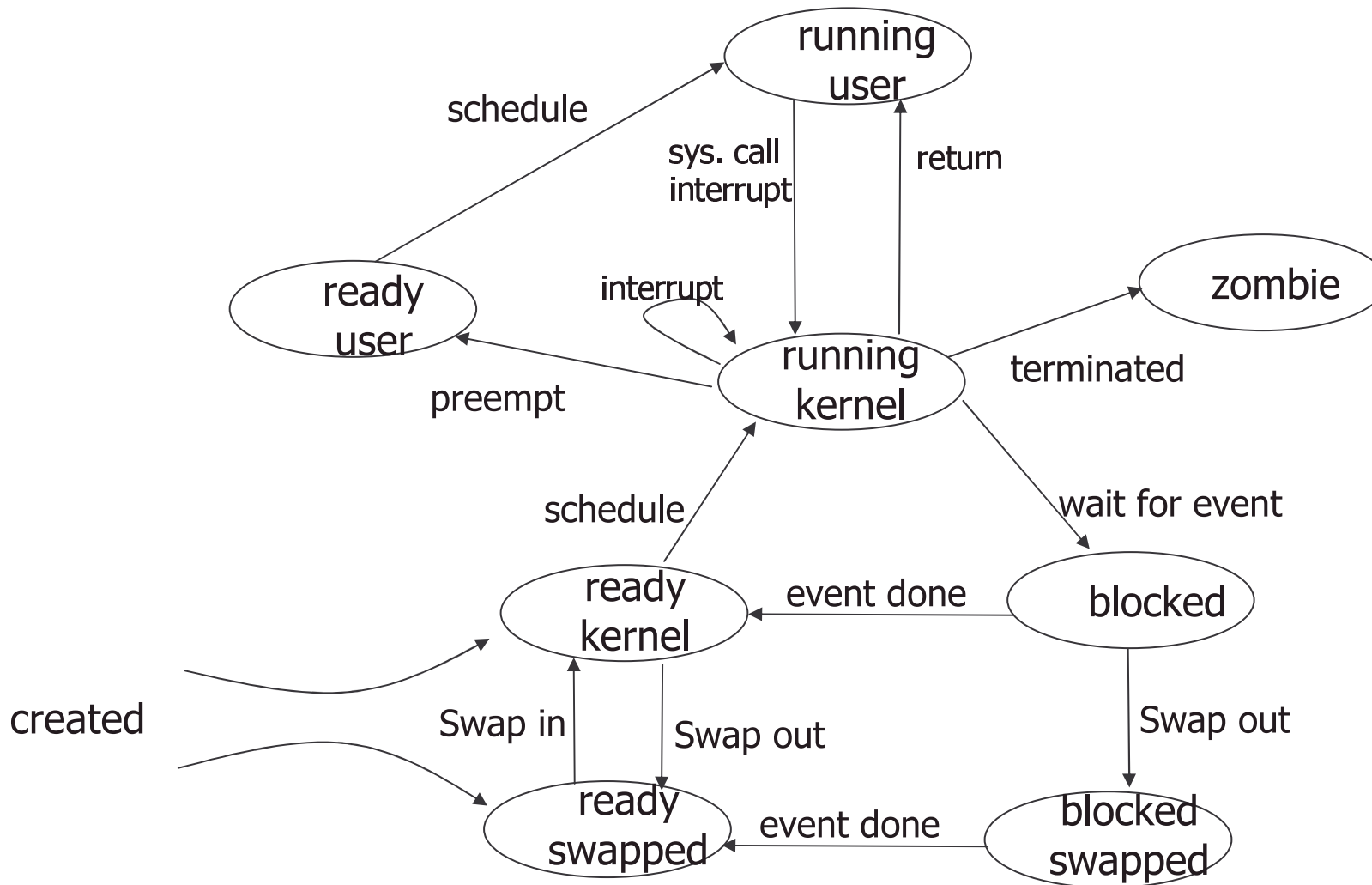
# Multiprogramming level

- Too many processes in memory
  - Thrashing, inability to run new processes
- The solution is swapping:
  - save all the resident set of a process to the disk (swapping out)
  - load the pages of another process instead (swapping in)
- Long-term and medium term scheduling decides which processes to swap in/out

# Long (medium) term scheduling

- Decision of which processes to swap out/in is based on
  - The CPU usage
  - Creating a balanced job mix with respect to I/O vs. CPU bound processes
- Two new process states:
  - Ready swapped
  - Blocked swapped

# UNIX process states



# Page size considerations

- Small page size
  - better approximates locality
  - large page tables
  - inefficient disk transfer
- Large page size
  - internal fragmentation
- Most modern architectures support a number of different page sizes
  - a configurable system parameter
  - Pentium processors support 4K or 4MB

# mlock(), munlock()

*int mlock(const void \*addr, size\_t len);*

*int munlock(const void \*addr, size\_t len);*

- mlock() disable paging for the memory in the given range
- All pages in the range are guaranteed to be in ram if the mlock() return successfully and stay there until munlock() is called
- Memory lockes do not stack
- Only root processes are allowed to lock pages
- Useful for real-time algorithms and high-security data processing

# /proc/meminfo

```
Tior@mos214:/proc> cat /proc/meminfo
          total:      used:      free:  shared: buffers:  cached:
Mem:  261423104 253939712  7483392          0 52805632 109502464
Swap: 271392768 111472640 159920128
MemTotal:      255296 kB
MemFree:       7308 kB
MemShared:     0 kB
Buffers:      51568 kB
Cached:       89728 kB
SwapCached:   17208 kB
Active:       53736 kB
Inactive:    160648 kB
HighTotal:    0 kB
HighFree:    0 kB
LowTotal:    255296 kB
LowFree:     7308 kB
SwapTotal:   265032 kB
SwapFree:    156172 kB
Tior@mos214:/proc>
```

# /proc/\$pid/status

```
Tior@mos214:/proc/17778> cat status
```

```
Name: csh
```

```
State: S (sleeping)
```

```
...
```

```
VmSize: 3264 kB
```

```
VmLck: 0 kB
```

```
VmRSS: 1824 kB
```

```
VmData: 916 kB
```

```
VmStk: 192 kB
```

```
VmExe: 268 kB
```

```
VmLib: 1576 kB
```

```
SigPnd: 0000000000000000
```

```
SigBlk: 0000000200000002
```

```
SigIgn: 000000000384004
```

```
SigCgt: 000000009812003
```

```
CapInh: 0000000000000000
```

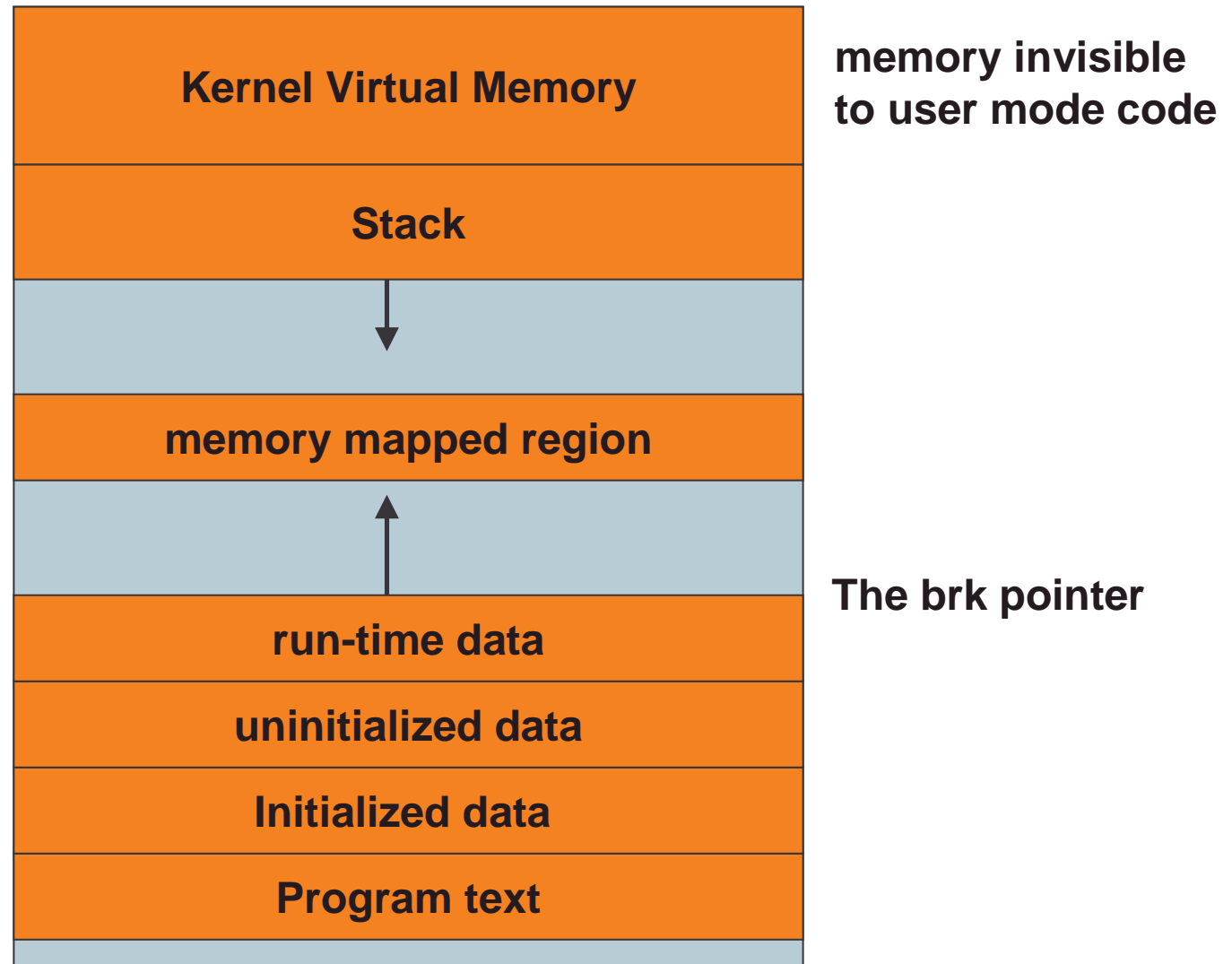
```
CapPrm: 0000000000000000
```

```
CapEff: 0000000000000000
```

# Mapping of Programs into Memory

- The pages of the binary file are mapped into regions of virtual memory
- Only when the program tries to access a given page a page fault will occur and the content of the file will be loaded to physical memory
- An ELF (**E**xecutable and **L**inking **F**ormat) format consists of a header followed by several paged-aligned regions
- The ELF loader read the header map the sections of the file to separate regions
- In a reserved region at one end of the address space sits the kernel (inaccessible to users)

# Memory layout of ELF programs



# Execution and Loading of User programs

- When `exec()` is called the kernel invokes the loader routine to load the content of the program file into physical memory
- There is no single routine for loading but a table of routines
- Linux support at least 2 formats of executables
  - The old a.out format
  - ELF format
- The command “*file*” can tell you the file type, including the binary format (if the file is binary)

# /proc/\$pid/maps

- This file contains the currently mapped memory regions and their access permissions

```
[root@mos214 proc]# cat 1/maps
```

address	perm	offset	dev	inode	
08048000-0804e000	r-xp	00000000	03:01	38859	/sbin/init
0804e000-08050000	rw-p	00005000	03:01	38859	/sbin/init
08050000-08054000	rwxp	00000000	00:00	0	
40000000-40016000	r-xp	00000000	03:01	71402	/lib/ld-2.2.4.so
40016000-40017000	rw-p	00015000	03:01	71402	/lib/ld-2.2.4.so
40017000-40018000	rwxp	00000000	00:00	0	
40031000-40168000	r-xp	00000000	03:01	116282	/lib/i686/libc- 2.2.4.so
40168000-4016d000	rw-p	00136000	03:01	116282	/lib/i686/libc- 2.2.4.so
4016d000-40172000	rw-p	00000000	00:00	0	
bffffe000-c0000000	rwxp	ffffff000	00:00	0	