CSCI 4061: Virtual Memory

Chris Kauffman

Last Updated:
Thu Dec 7 12:52:03 CST 2017

Logistics: End Game

Date		Lecture	Outside
Mon	12/04		Lab 13: Sockets
Tue	12/05	Sockets	
Thu	12/07	Virtual Memory	
Mon	12/11		Lab 14: Review
Tue	12/12	Review	P5 Due
Wed	12/13	Classes End	
Wed	12/20	10:30am-12:30pm	Final Exam

Reading

- Stevens/Rago: Ch 16 Sockets
- Virtual Memory Reference: Bryant/O'Hallaron, Computer Systems. Ch 9 (CSCI 2021)
- `mmap()`: Linux System Programming, 2nd Edition By: Robert Love (library site link)

Goals: Finish Sockets

Lab13: Client Sockets

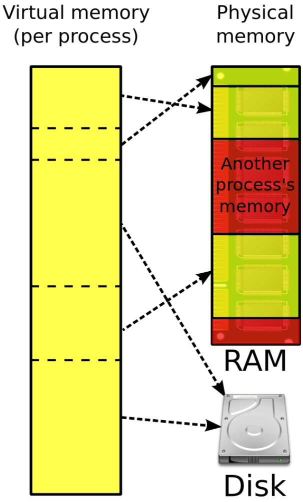
How did it go?

Project 2

Updates and Questions

Addresses are a Lie

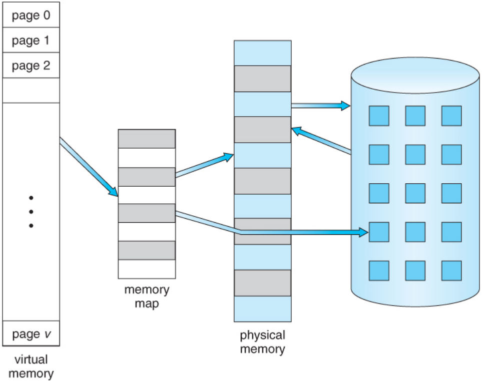
- Operating system uses tables and hardware to translate every program address
- Processes know virtual addresses which are translated via the memory subsystem to physical addresses in RAM and on disk
- Contiguous virtual addresses may be spread all over physical memory



Source: WikiP Virtual Memory

Address Translation

- OS maintains tables to translate virtual to physical addresses
- This needs to be FAST so usually involves hardware: Memory Manager Unit (MMU) and Translation Lookaside Buffer (TLB)
- Address translation is NOT CONSTANT O(1), has an impact on performance of real algorithms*



Source: John T. Bell Operating Systems Course Notes

*See: On a Model of Virtual Address Translation (2015)

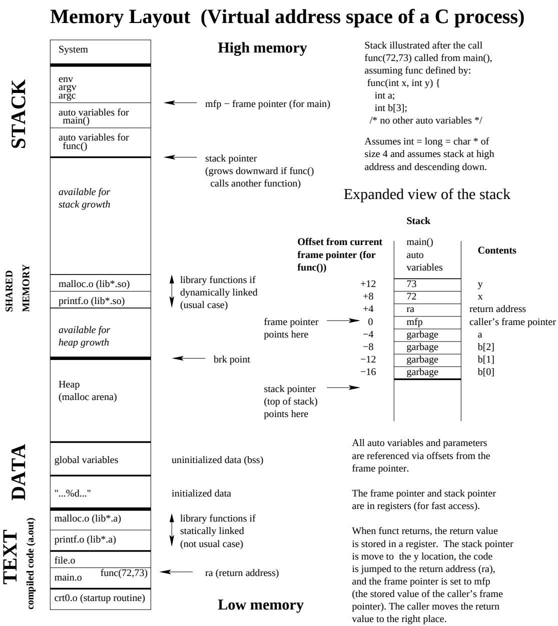
Pages and Mapping

- Memory is segmented into hunks called pages, 4Kb is common (use `page-size.c` to see your system's page size)
- OS maintains tables of which pages of memory exist in RAM, which are on disk
- OS maintains tables per process that translate process virtual addresses to physical pages
- Shared Memory can be arranged by mapping virtual addresses for two processes to the same memory page

Proc	VirtPage	PhysPage	
123	0	1046	Shared
	1	900	
	2	2032	
456	0	800	
	1	400	
	2	1046	Shared
	3	3040	

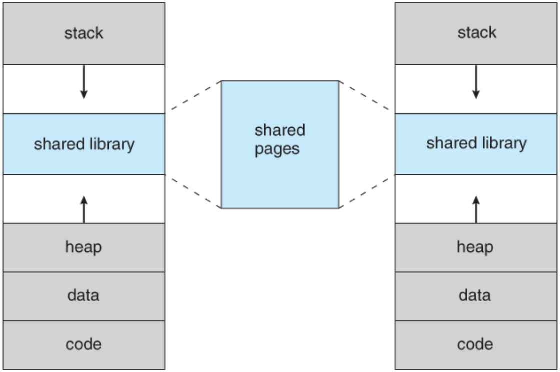
Exercise: Process Memory Image and Libraries

- How many programs on the system need to use `malloc()` and `printf()`?
- Where is the code for `malloc()` or `printf()` in the process memory?



Shared Libraries: *.so Files

- Code for libraries can be shared
- `libc.so`: shared library with `malloc()`, `printf()` etc in it
- OS puts into one page, maps all linked procs to it



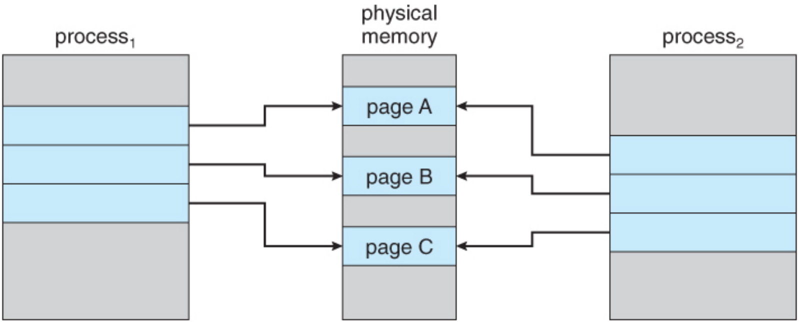
Source: John T. Bell Operating Systems Course Notes

Exercise: Recall `fork()`

- What does `fork()` do?
- What does the result of a `fork()` look like?
- What *seems* to need to happen for this to work

Fork and Shared Pages

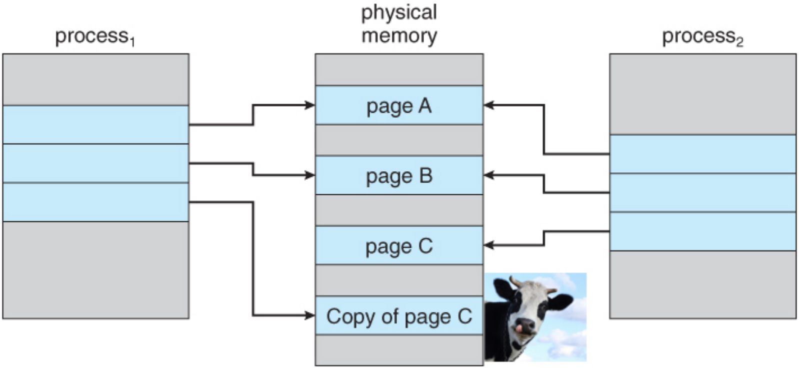
- `fork()`'ing a process creates a nearly identical copy of a process
- Might need to copy all memory form parent to child pages
- Can save a lot of time if memory pages of child process are shared with parent - no copying needed (initially)
- What's the major danger here?



Source: John T. Bell Operating Systems Course Notes

Fork, Shared Pages, Copy on Write (COW Pages)

- If neither process writes to the page, sharing doesn't matter
- If either process writes, OS will make a copy and remap addresses to copy so it is exclusive
- Fast if hardware Memory Management Unit and OS know what they are doing (Linux + Parallel Python/R + Big Data)



Source: John T. Bell Operating Systems Course Notes

Shared Memory

Most Unix Systems provide System V and POSIX means for a program to explicitly create shared memory.

```
// SYSTEM V SHARED MEMORY
int shmget(key_t key, size_t size, int shmflg);
// create/acquire a segment of shared memory assocaited with given key
// and size, returns id assocaited with segment

void *shmat(int shmid, const void *shmaddr, int shmflg);
// attach to shared memory with given id, return address of shared
// memory, may specify preferred address or NULL

// POSIX SHARED MEMORY
int shm_open(const char *name, int oflag, mode_t mode);
// get an id (file descriptor) for segment of shared memory, similar
// to open() system call but memory only

void *mmap(void *addr, size_t len, int prot, int flags,
           int fd, off_t off);
// map given file descriptor to a memory address. Reads/writes
// associated with address are reflected into the contents of the file
// descriptor potentially resulting in reads/writes to backing files.
```

`mmap()`: Mapping Addresses is Ammazing

- `ptr = mmap(NULL, size,...,fd,0)` arranges backing entity of `fd` to be mapped to be mapped to `ptr`
- `fd` might be shared memory created with `shm_open()`
- `fd` might be a file opened with `open()`...
  - Wait, what?

```
int fd = open("gettysburg.txt", O_RDONLY);
// open file to get file descriptor

char *file_chars = mmap(NULL, size, PROT_READ, MAP_SHARED,
                        fd, 0);
// pointer to file contents call mmap with given size and file
// descriptor read only, potentially share, offset 0

printf("%c",file_chars[0]);                // print 0th char
printf("%c",file_chars[5]);                // print 5th char
```

Exercise: Examine `mmap-demo.c`

- Determine what it does
- Are there any limits to the information that is produced by the program
- How might one modify the program to accommodate arbitrarily sized files?
- Answer in `mmap-print-file.c`

`mmap()` allows file reads/writes without `read()`/`write()`

- Memory mapped files are not just for reading
- With appropriate options, writing is also possible

```
char *file_chars =
  mmap(NULL, size, PROT_READ | PROT_WRITE,
       MAP_SHARED, fd, 0);
```

- Amazing stuff: assign to memory, OS reflects change into the file
- Example: `mmap-tr.c` to transform one character to another

`mmap()` Flexibility is complete

- `mmap()` just gives a pointer: can assert that it points to binary data like `structs` as well
- See example: `mmap-specific-stock.c` for an example of this
- Multiple processes can map files to shared memory to communicate, read/write same files, cooperate
- IPC control mechanisms such as semaphores, message queues, mutexes should be used to control shared files to prevent read/write conflicts

`mmap()` Comparisons

Benefits

- Avoid `read()` into memory, change, `write()` cycle
- Saves memory and time
- Many Linux mechanisms backed by `mmap()` like shared memory

Drawbacks

- Always maps pages of memory $\sim$ 4096b (4K)
- For small maps, lots of wasted space
- No bounds checking, just like everything else in C