Exploring Processes

Nimish Mishra
@manwe sulimo

Agenda

Program vs Process

Aren’t they the same?

Process control block

What OS adds to a process

/proc

And knowing what goes under the hood

Appendix

Some extra things:

-Foreground / Background processes
-Symlinks

Questions

-Technical?
-Non-technical?

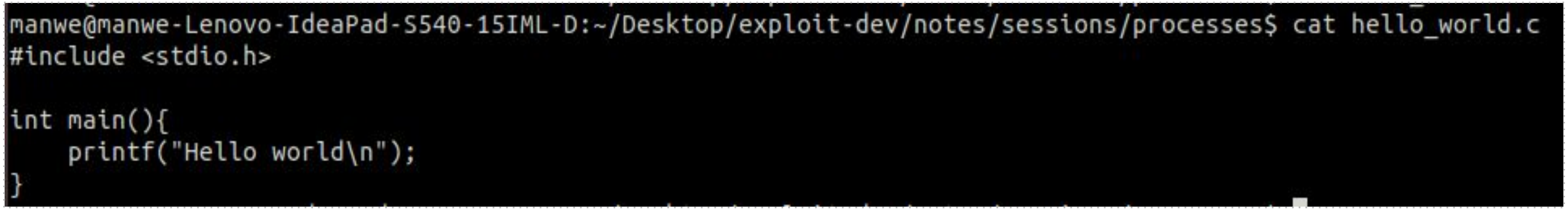
- How does a program look like?
- How does a process look like?
- Delving into the Process Control Block

Program vs Processes

Process Control Block

How does a program look like?

- Like a normal program you write in a programming language like C



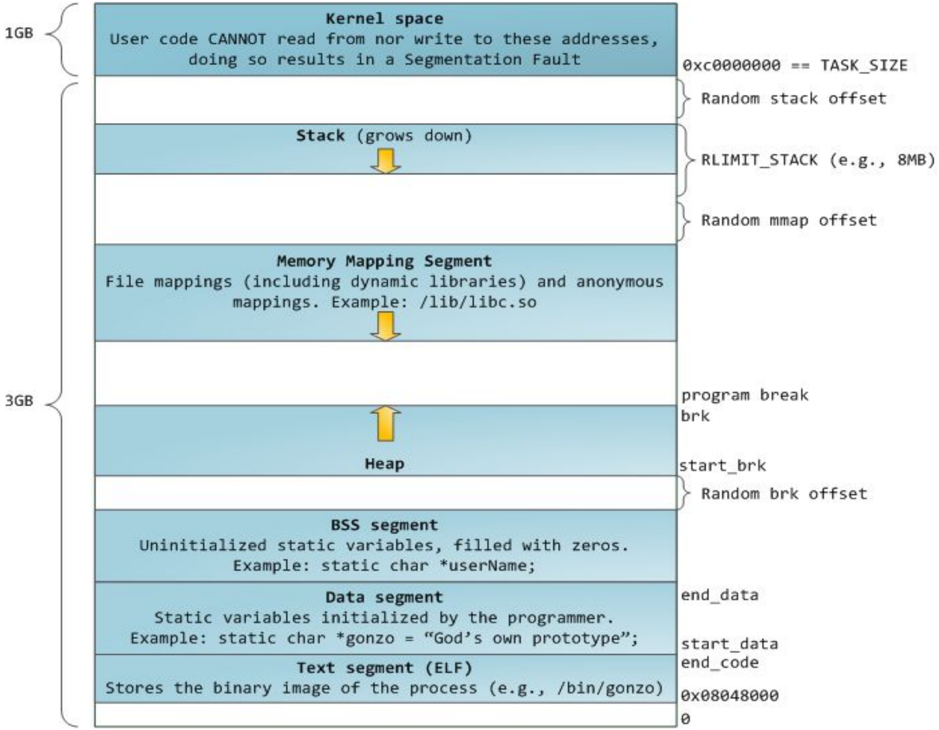
```
manwe@manwe-Lenovo-IdeaPad-S540-15IML-D:~/Desktop/exploit-dev/notes/sessions/processes$ cat hello_world.c
#include <stdio.h>

int main(){
    printf("Hello world\n");
}
```

- Can this run directly on your processor?
  - No...
- What does it need to run on your processor?
  - Several bookkeeping information like where to store data, where to read files from, how much stack to use, how long to run etc etc.
- And who gives all this information?
  - Operating system

How does a process look like?

- Process = program + extra information that operating system provides
- Do a `pstree` to know the process tree of your system (can you add yours?)



Kernel space
User code CANNOT read from nor write to these addresses, doing so results in a Segmentation Fault

Stack (grows down)

Memory Mapping Segment
File mappings (including dynamic libraries) and anonymous mappings. Example: /lib/libc.so

Heap

BSS segment
Uninitialized static variables, filled with zeros.
Example: static char *userName;

Data segment
Static variables initialized by the programmer.
Example: static char *gonzo = "God's own prototype";

Text segment (ELF)
Stores the binary image of the process (e.g., /bin/gonzo)

Pointer to the process parent	Process State
Pointer to the process child	

Process Identification Number

Process Priority

Program Counter

Registers

Pointers to Process Memory

Memory Limits

List of open Files

. . .

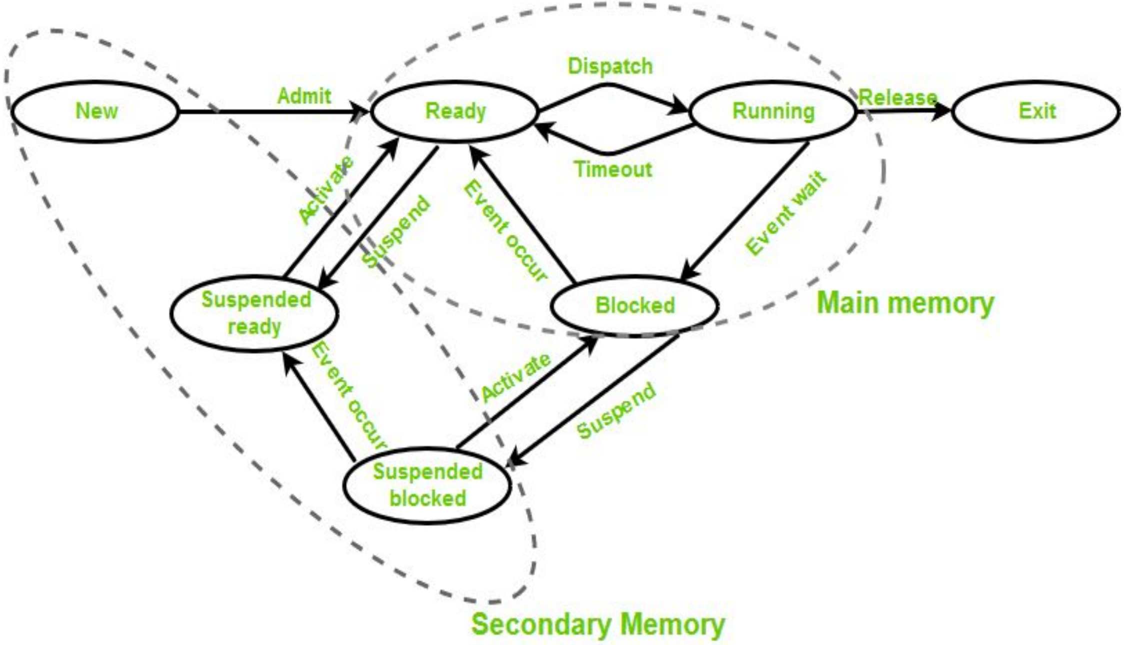
Delving into the Process Control Block (PCB)

- True multitasking systems *swap* processes in and out of the processor
- OSes use PCBs to *shift* processes from one state to another
- context switch: Switch from **running to ready state**

Note how main memory and secondary memory participate in state diagram.

- New: program → process
- Ready: Ready to run
- Running: Running
- Blocked: Waiting (ex. for I/O)
- Exit: Done :) Destroy the PCB

Suspend is done to free main memory for more PCBs.

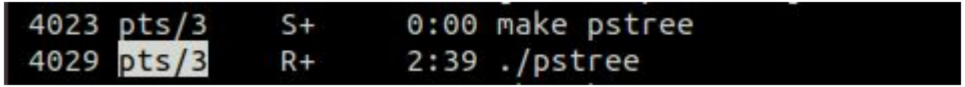


/proc

- /proc filesystem
- What we can learn from /proc

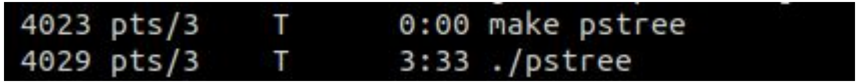
/proc filesystem

- Virtual filesystem created at boot and destroyed during shutdown
- Serves as an interface between kernel and userspace
- Possibly dangerous? For some reason, MacOS decided to NOT do /proc
- To access the /proc filesystem
  - Create a process that runs indefinitely (and take your time to explore :))
  - Let’s say process is named **pstree**. Do a **ps ax | grep pstree** to get the PID (ex. **3991**)
  - Find interesting stuff at **cd /proc/3991**
- How does a **ps ax** output look like



```
4023 pts/3    S+     0:00 make pstree
4029 pts/3    R+     2:39 ./pstree
```

Output while the process is running



```
4023 pts/3    T      0:00 make pstree
4029 pts/3    T      3:33 ./pstree
```

Output when process stopped with Ctrl+Z

S: Sleep (waiting for something to complete)
R: Running or ready
+: a foreground process

T: Stopped/suspended (but not terminated)

More process state codes: https://linux.die.net/man/1/ps

What can we learn from /proc

- Process state: **cat /proc/<PID>/status**
  - See current run state, voluntary/non-voluntary context switches, CPUs allowed list
- Process memory maps: **cat /proc/<PID>/maps**
  - See ranges of heap and stack memory
- Process environment: **cat /proc/<PID>/environ**
- Process I/O: **cat /proc/<PID>/io**
  - See characters read and written. Verify length of messages to be reflected
- Process pagemaps: write a program to access process pagemaps
- Process limitations: **cat /proc/<PID>/limits**
  - Upper bound on several process parameters
- Process file descriptors: **ls -la /proc/<PID>/fd/**
  - Verify symlinks

- Foreground/background processes
- Symlinks

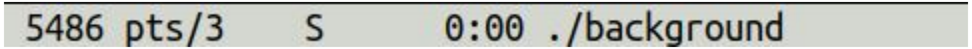
Appendix

Foreground/background processes

- Append a **&** to run a program in the background

- Do a **fg** to bring it in foreground

- What's the difference between a foreground and background process?



```
5486 pts/3    S      0:00 ./background
```

Process state: S

Symlinks

- File pointers
- Two kinds of links: **hardlinks** and **softlinks**
- **Hardlinks**: are duplicate copies (means they remain after original file is deleted). Create through **ln source target**
- **Softlinks:** are pointers (if original is deleted, they are also corrupted). Create through **ln -s source target**
- Usually appear as → in **ls -la** output
- Verify:
  - **For hard links**
    - They are copies
    - change in original done in copy too
    - delete original but hard link remains
  - **For soft links**
    - They are not copies. Changes in original are reflected in the symlink too
    - Deleting original corrupts the symlink

Link to everything:
https://github.com/NimishMishra/exploit-dev/tree/master/notes/sessions/processes

Questions?