Systems Programming

Lecture 05 - Process, Memory

Overview

Last Time

- Stat System Call
- Introduction to Raspberry Pi

Readings for today

- Chapter 6 – Processes
- Chapter 7 – Memory
- Chapter 8 – User and groups

Prologue

- Processes and memory

Epilogue

- Users and groups

Chapter 6 Processes

- 6.1 Processes and Programs
- 6.2 Process ID and Parent Process ID
- 6.3 Memory Layout of a Process
- 6.4 Virtual Memory Management
- 6.6 Command-Line Arguments (*argc*, *argv*)
- 6.7 Environment List

6.1 Processes and Programs

- A process is an instance of an executing program. In this section, we elaborate on this definition and clarify the distinction between a program and a process.
- A program is a file containing a range of information that describes how to construct a process at run time.
- Process : an entity defined by kernel to which system resources are allocated to execute a program

Information resides in a program

- **Binary format identification:**
  - a.out – “assembler output”
  - COFF : Common Object File Format
  - ELF : Executable and Linkable Format
- **Machine language**
  - od –c
  - od –d
- **Program entry point**
- **Data :** variables
- **Symbol and relocatable tables :** var/func locations
- **Shared-libraries**

Process ID and Parent Process ID

SYNOPSIS

```
    #include <sys/types.h>
    #include <unistd.h>

    pid_t getpid(void);
    pid_t getppid(void);
```

DESCRIPTION

getpid() returns the process ID of the calling process. (often used to generate unique temp filenames

getppid() returns the process ID of the parent of the calling process.

- Useful to kill() a process

Pid_t

- pid_t limits to 32,767
- on 64-bit platforms, it can be adjusted to any value up to 2^{22}

- ID counter resets to 300, Why?

- `/proc/PID/status`

Memory Layout of a Process

-

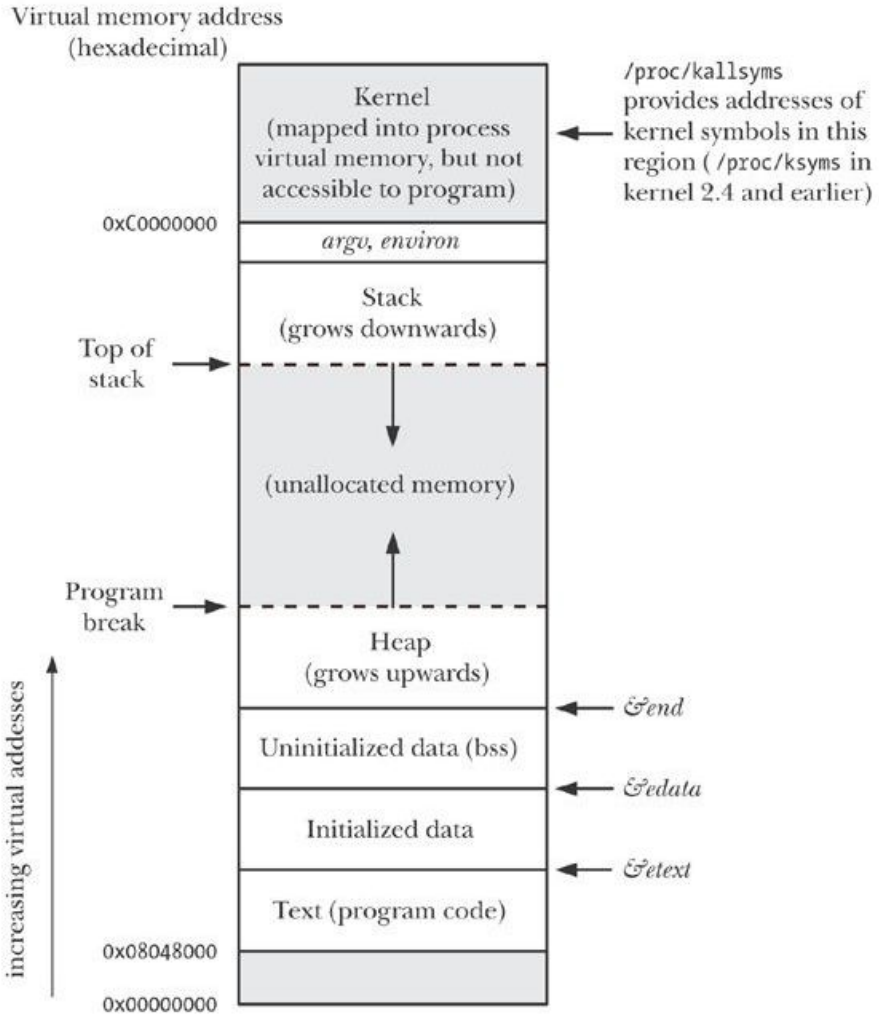


Figure 6-1. Typical memory layout of a process on Linux/x86-32

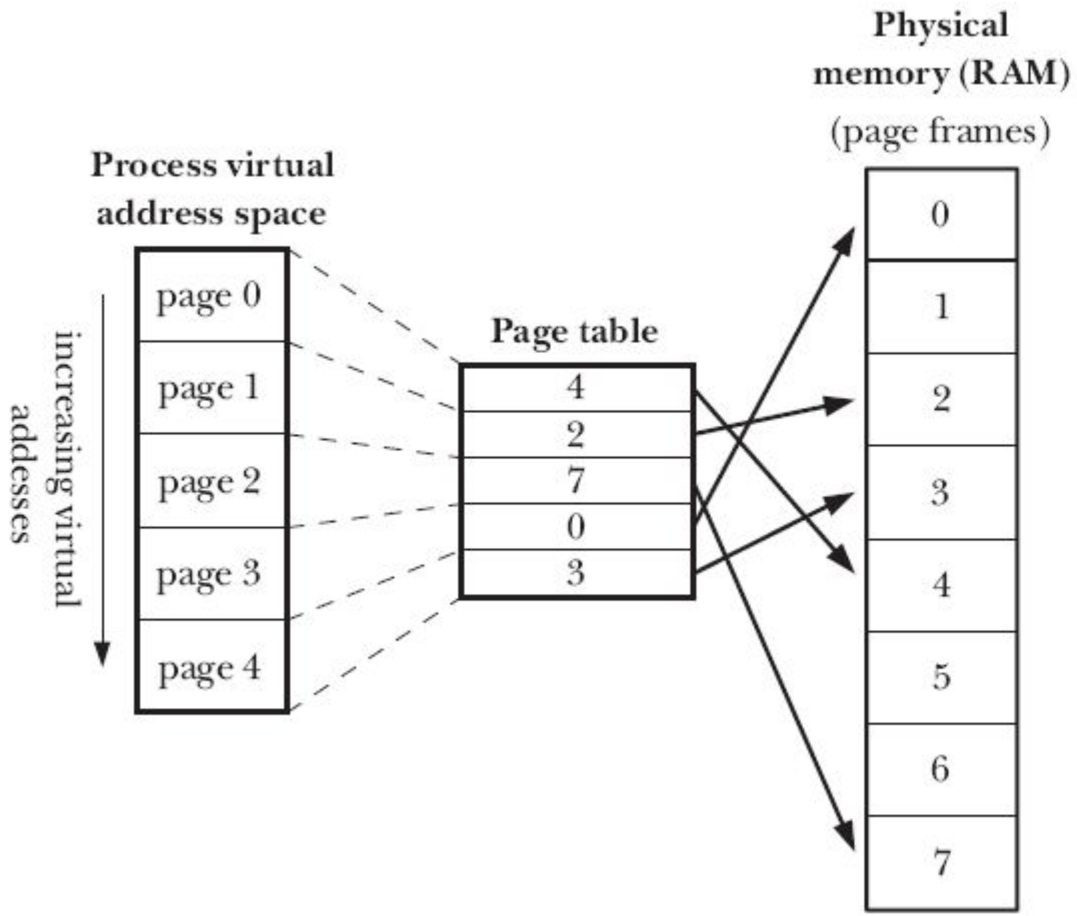
Virtual memory

- Useful in `fork()`
- Shared memory, pipes
- Swap area
- Page fault
- On x86-32, pages are 4096 bytes in size.
- sysconf(_SC_PAGESIZE),

Virtual memory

- Linux employs a technique known as virtual memory management.
- The aim of this technique is to make efficient use of both the CPU and RAM (physical memory) by exploiting a property that is typical of most programs: locality of reference

Overview of Virtual Memory



Process virtual address space

page 0

page 1

page 2

page 3

page 4

increasing virtual addesses

Page table

4

2

7

0

3

Physical memory (RAM) (page frames)

0

1

2

3

4

5

6

7

Command line arguments

- argv[argc] is NULL, what about argv[0] ?

- “gzip(1), gunzip(1), and zcat(1) commands, all of which are links to the same executable file”

Csce510-001/TPLI/proc

```
root> pwd
/class/csce510-001/TLPI/proc
root> ls *.c
bad_longjmp.c
longjmp.c
modify_env.c
setenv.c
t_getenv.c
display_env.c
mem_segments.c
necho.c
setjmp_vars.c
```

necho.c - look familiar

```
#include "tlpi_hdr.h"

int
main(int argc, char *argv[])
{
    int j;

    for (j = 0; j < argc; j++)
        printf("argv[%d] = %s\n", j,
argv[j]);

    exit(EXIT_SUCCESS);
}
```

Pointers -argv again

```
char **p;    /* ... 6.1 ... */
for (p = argv; *p != NULL; p++)
   puts(*p);
```

- Alternative way to access : /proc/PID/cmdline
- <2.6.23 : ARG_MAX - sysconf()
  - `/usr/include/limits.h`
- “the limit on the total space used for argv and environ can be controlled via the RLIMIT_STACK” (>= 2.6.23)
- `man execve`

The Environment

- env - list of name=value strings
  - Environment variables
  - env | fgrep SHELL
  - HOME, PATH
- Create a shell variable
- c5=/classsysprog2013
- export c5 -- puts it into the environment
- Inside your program :
  - setenv, unsetenv, clearenv

env vs printenv

- man env
- man printenv
- man -s7 environ

Accessing environment from C

- char **environ;   **// global variable**

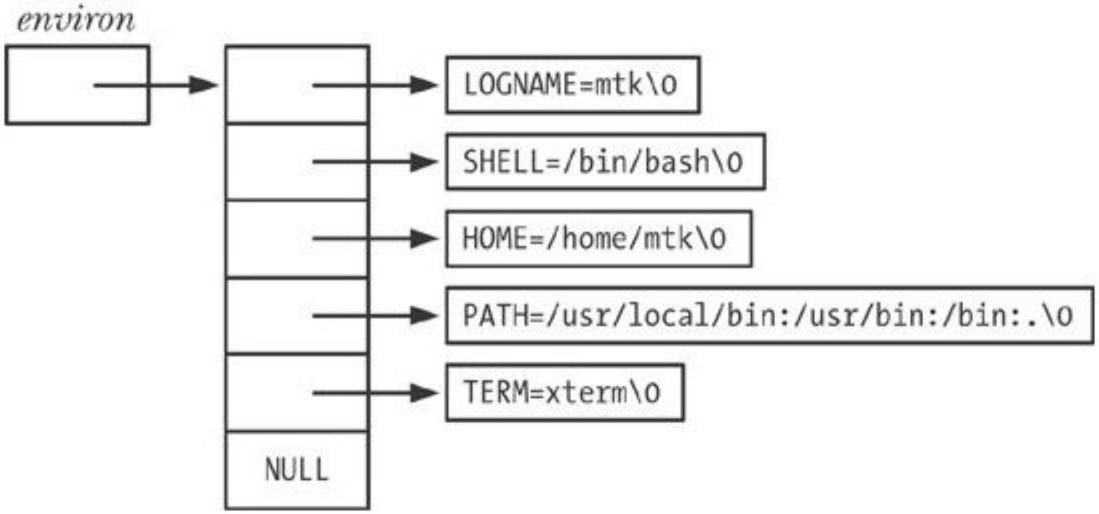


Figure 6-5. Example of process environment list data structures

- **less** TLPI/proc/display_env.c
- **Alternatively :** int main(int argc, char *argv[], char ***envp**[])

Accessing environment from C

```
char *getenv(const char *name);
```

DESCRIPTION - The getenv() function searches the environment list to find the environment variable name, and returns a pointer to the corresponding value string.

PUTENV(3) Linux Programmer's Manual PUTENV(3)

NAME

putenv - change or add an environment variable

SYNOPSIS

```
#include <stdlib.h>

int putenv(char *string);
```

Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

putenv(): _SVID_SOURCE || _XOPEN_SOURCE

DESCRIPTION The putenv() function adds or changes the value of environment variables. The argument string is of the form name=value.

NAME

setenv - change or add an environment variable

SYNOPSIS

```
#include <stdlib.h>

 int setenv(const char *name,
const char *value, int overwrite);

 int unsetenv(const char *name);
```

“On occasion, it is useful to erase the entire environment, and then rebuild it with selected values. For example, we might do this in order to execute set-user-ID programs in a secure manner (Don't Trust Inputs or the Environment).

We can completely erase the environment by :

```
environ = NULL;
```

or

```
int clearenv(void);
```

Chapter 7 MEMORY ALLOCATION

- 7.1 Allocating Memory on the Heap
- 7.1.1 Adjusting the Program Break: *brk()* and *sbrk()*
- 7.1.2 Allocating Memory on the Heap: *malloc()* and *free()*
- 7.1.3 Implementation of *malloc()* and *free()*
- 7.1.4 Other Methods of Allocating Memory on the Heap
- 7.2 Allocating Memory on the Stack: *alloca()*
- 7.3 Summary
- 7.4 Exercises

brk() and sbrk()

NAME -- brk, sbrk - change data segment size

SYNOPSIS

```
#include <unistd.h>
int brk(void *addr);
void *sbrk(intptr_t increment);
brk(), sbrk(): _BSD_SOURCE || _SVID_SOURCE || _XOPEN_SOURCE >= 500
```

DESCRIPTION - brk() and sbrk() change the location of the program break, which defines the end of the process's data segment. Increasing the program break has the effect of allocating memory to the process; decreasing the break deallocates memory.

- brk() sets the end of the data segment to the value specified by *addr*, IF that value is reasonable, the system has enough memory, and the process does not exceed its maximum data size; RLIMIT_DATA - see *setrlimit(2)*
- sbrk() increments the program's data space by increment bytes.

Allocating on the Heap

- **malloc, free, etc.**

```
#include <stdlib.h>

 void *calloc(size_t nmemb, size_t size);
 void *malloc(size_t size);
 void free(void *ptr);
 void *realloc(void *ptr, size_t size);
```

- **TLPI/memalloc/free_and_sbrk.c**

Implementation of malloc and free

- Malloc() tries to find in *free-list*, otherwise calls sbrk( ) (with some extras)
- Returned block :

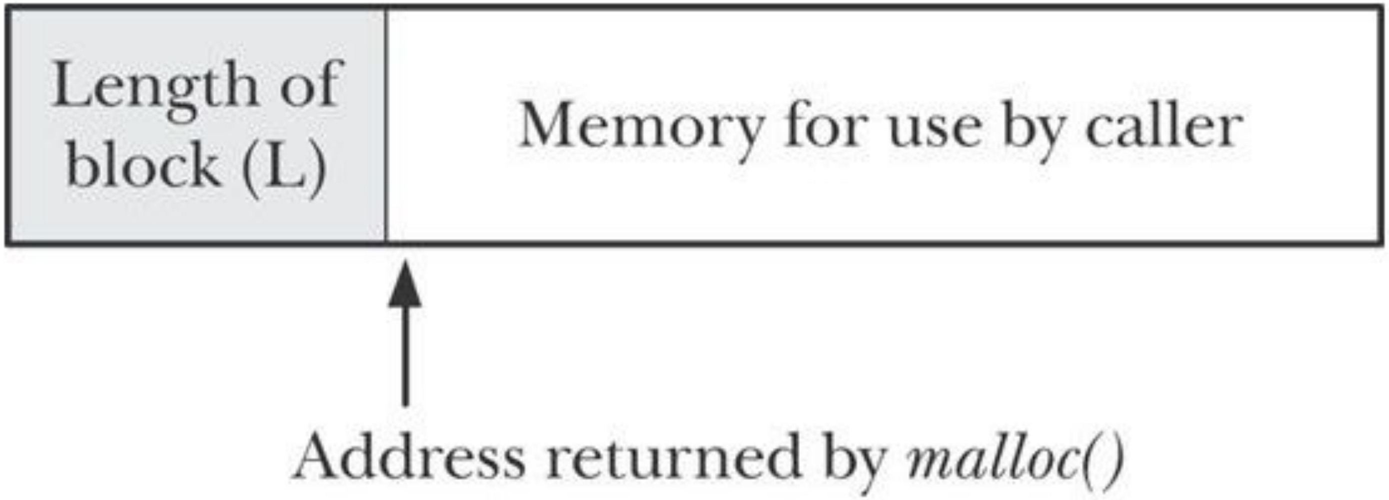


Figure 7-1. Memory block returned by malloc()

Freeing the block allocation

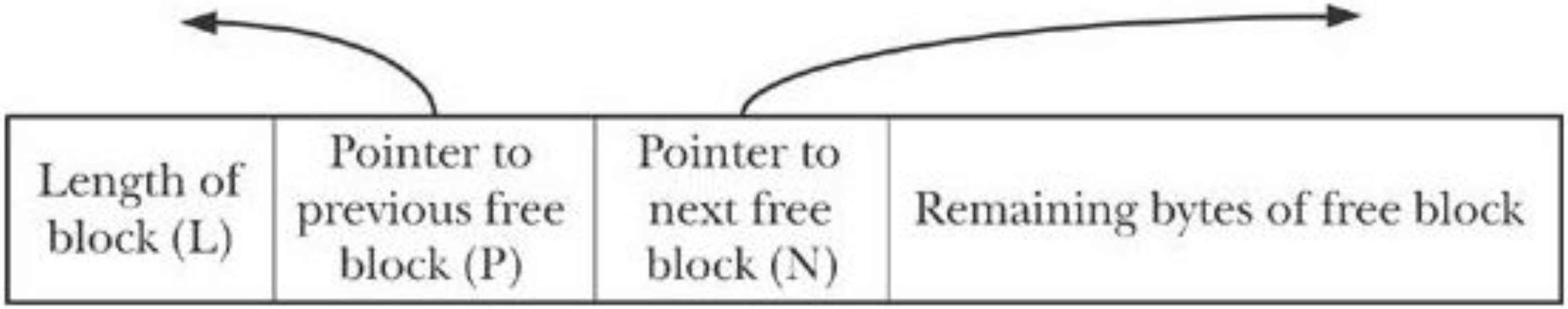

Length of block (L)
Pointer to previous free block (P)
Pointer to next free block (N)
Remaining bytes of free block

Figure 7-2. A block on the free list

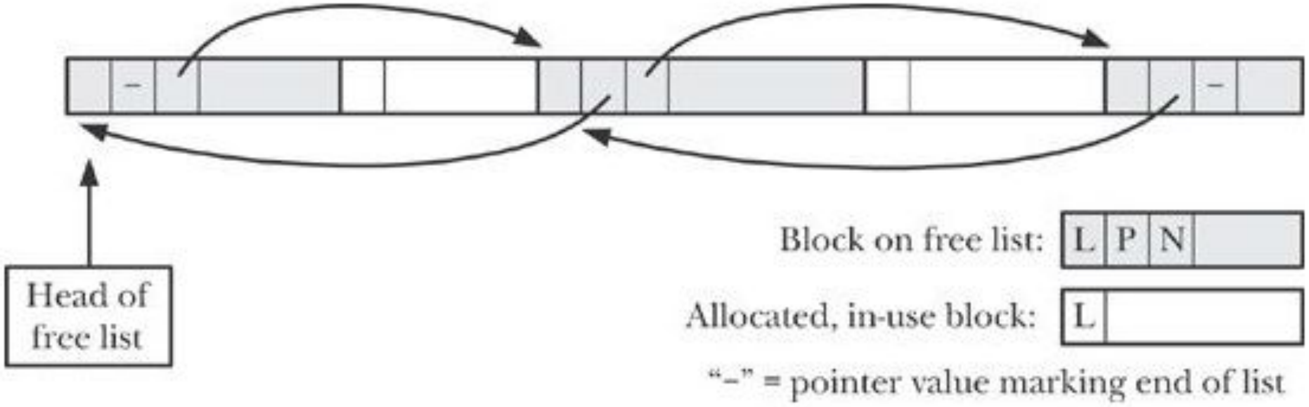


Figure 7-3. Heap containing allocated blocks and a free list

Issues : Memory leaks

- **Memory leaks**
  - Reached the limit of available virtual memory
  - Fail to allocate memory
  - Example :
    - not `free()`-ing your malloc
    - you touch bytes outside your allocation

- **Checking for memory leaks?**
  - glibc : mtrace(), mcheck(), ….
  - external libs : valgrind, ….

Chapter 8 USERS AND GROUPS

- 8.1 The Password File: /etc/passwd
- 8.2 The Shadow Password File: /etc/shadow
- 8.3 The Group File: /etc/group
- 8.4 Retrieving User and Group Information
- 8.5 Password Encryption and User Authentication
- 8.6 Summary
- 8.7 Exercises

Chapter 9 PROCESS CREDENTIALS

- 9.1 Real User ID and Real Group ID
- 9.2 Effective User ID and Effective Group ID
- 9.3 Set-User-ID and Set-Group-ID Programs
- 9.4 Saved Set-User-ID and Saved Set-Group-ID
- 9.5 File-System User ID and File-System Group ID
- 9.6 Supplementary Group IDs
- 9.7 Retrieving and Modifying Process Credentials
- 9.7.1 Retrieving and Modifying Real, Effective, and Saved Set IDs
- 9.7.2 Retrieving and Modifying File-System IDs
- 9.7.3 Retrieving and Modifying Supplementary Group IDs
- 9.7.4 Summary of Calls for Modifying Process Credentials
- 9.7.5 Example: Displaying Process Credentials
- 9.8 Summary
- 9.9 Exercises

QA

Setjmp / longjmp

NAME

setjmp, sigsetjmp - save stack context for non-local goto

SYNOPSIS

```
#include <setjmp.h>

int setjmp(jmp_buf env);

int sigsetjmp(sigjmp_buf env, int savesigs);
```

LONGJMP(3)   Linux Programmer's Manual   LONGJMP(3)

NAME

longjmp, siglongjmp - non-local jump to a saved stack context

SYNOPSIS

```
#include <setjmp.h>

void longjmp(jmp_buf env, int val);

void siglongjmp(sigjmp_buf env, int val);
```