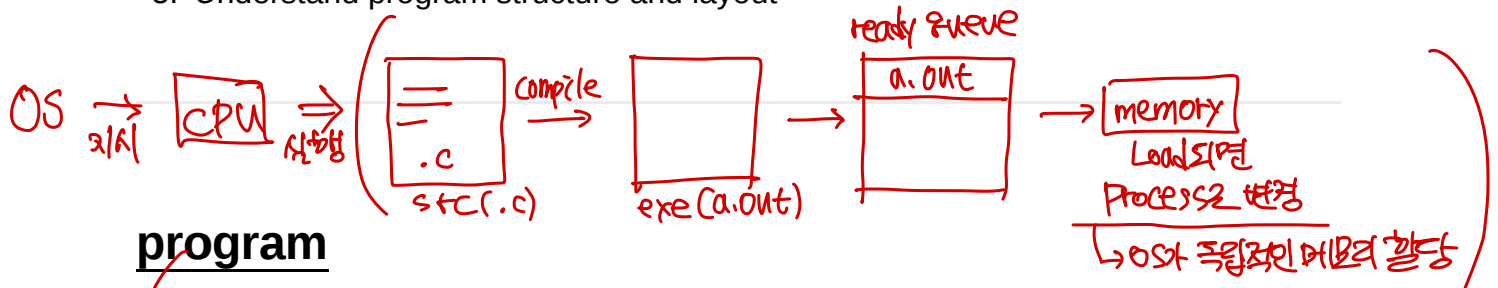


chapter02. Programs, Processes and Threads

Objectives

1. Learn about programs, processes and threads
2. Experiment with memory allocation and manipulation
3. Explore implications of static objects
4. Use environment variables for context
5. Understand program structure and layout

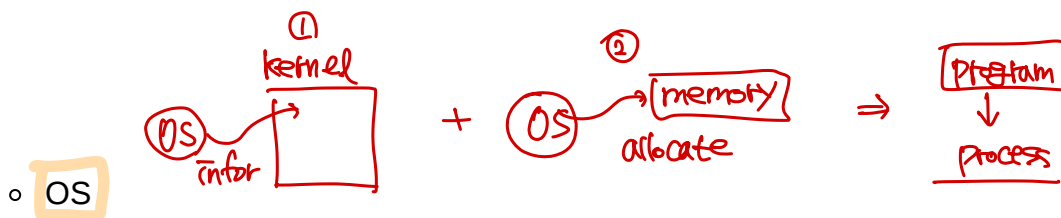


program

- A prepared sequence of instructions to accomplish a defined task
 - compiler
 - ① translates each source file into an object file.
 - ② and links the individual object files with the necessary libraries to produce an executable module.

process

- An instance of a program that is executing
 - CPU : program → process 과정을 OS가 하도록 지시
 - .c source → a.out exe → (ready queue) → memory load → process



- ① ■ add the appropriate information(PID, state , etc.) in the kernel data structures
- ② ■ allocate the necessary resources to run the program code
다하면
⇒ the program has become a process

- Process has an address space / and at least one flow of control ⇒ thread
주소 공간 적어도 한 개의 흐름, 즉 thread

Threads and thread of execution

Program counter : CPU 내부에 있는 register ⇒ 명령어 pointer

실행할 process의 address 가지고 있음

- to determine which process instruction is executed next

A thread of execution

- the stream of instructions "P.C가 할당된 주소"
- represented by the sequence of instruction addresses (assigned to the program counter.)
- example



- process1 : 245, 246, 247 ... / process2 : 10, 11, 12 ...
- CPU : context-switch instruction ⇒ 동시에 실행하는 것처럼 보임
 - result : 245, 246, 247, 245, 246 [context-switch instruction] 10, 11, 12, 13 [context-switch instruction] 247, 245, 249

Multiple threads

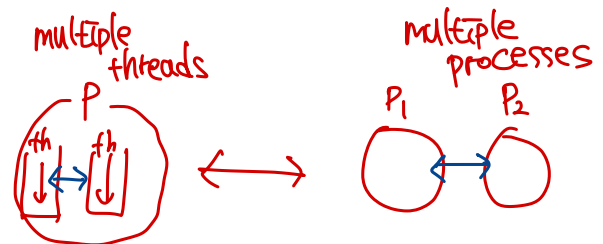
Thread

process의 subset, 여러 개의 실행 흐름 → 정보의 양이 적음

- An absolute data type that represents a thread of execution within a process
- its own execution task, program counter, register set and state.

Multiple processes vs. multiple threads

- multiple threads : 낮은 overhead parallelism 제공 (병렬적으로 실행함에 있어 추가적인 추가 시간)
 - OS의 도움 필요 x \Rightarrow 공유하는 메모리가 존재
 - \triangle but thread 간의 동기화 memory 생각을 해야 함 \Rightarrow 같은 process 주소와 resource를 사용
 - avoid context switches and allow sharing of code and data
- multiple processes
 - OS의 도움 필요 o

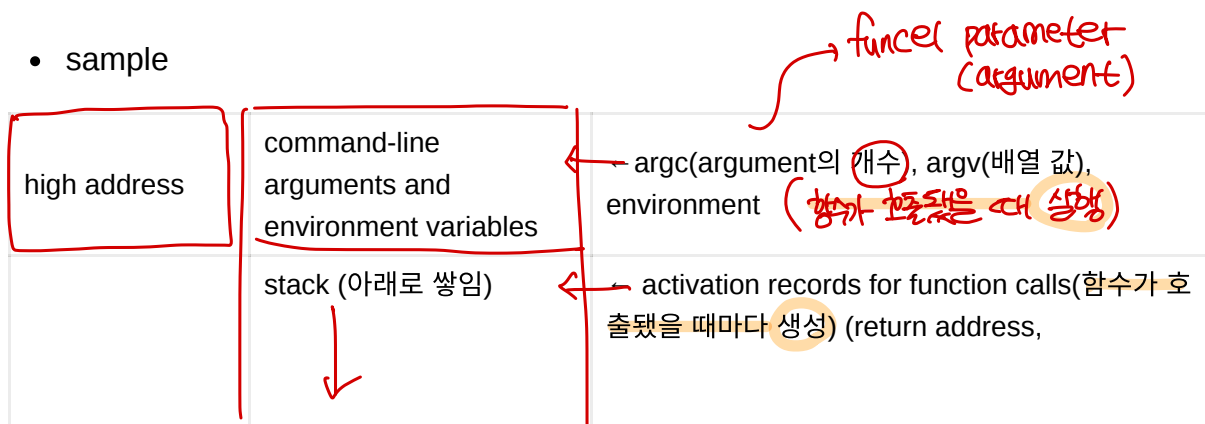


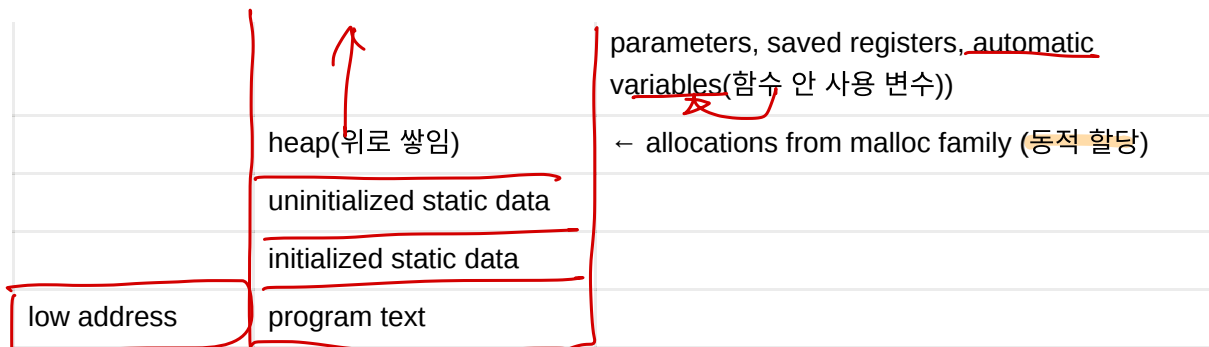
Layout of a program image

A program image

- A contiguous(연속적인) block of logical memory occupied by the program executable
- Has several distinct sections
 - ① The program text is shown in low-order memory. 내장지수
 - ② The initialized and uninitialized static variables have their own sections.
 - ③ stack and heap.

- sample





Activation record (활동 기록)

- A block of memory allocated on the top of the process stack to hold the execution context of a function during a call.
- Each function call creates a new activation record.
- Contains the return address, the parameters, status information, a copy of some of the CPU register values, automatic variables.

⇒ function이 return 할 때 stack에서 지워짐.

Library function calls

- SYNOPSIS box(간단한 man page)
 - A condensed version of function specification (ex. man pages for complete information
 - Gives required header files and the function prototypes.

Error

- Traditional UNIX functions usually return -1(or NULL) / set errno

- 하지만, 모든 새로운 함수마다 에러 처리가 다름

⇒ errno를 사용하지 않고 function return 값을 error number 직접 return하여 사용하기도 함.

- Error handling functions

1) void perror(const char *x) → string

① 사용자가 입력한 str 출력 ② errno 출력(str로)

- #include <stdio.h>

→ output. (errno에 해당하는 error message 출력)

2) char* strerror(int errnum)

- parameter로 넣어온 errno에 해당하는 error message 출력

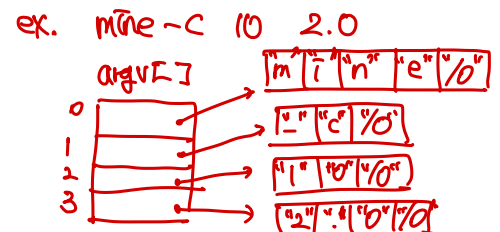
- `#include <string.h>`
- `return` : a pointer to the system error message corresponding to the `errno`
- 만약 `errno` 바뀌면 저장 후 다시 사용할 필요가 있을 때 `restore` 해야 함.
- 만약 함수 자체가 error 면 중첩 가능하기에 `restore` 해야 함.
stack for C)

• Good model of a function

- ① Do not exit from functions, instead, return an error value
- ② Do not make unnecessary assumptions about sizes of buffers
- ③ Use standard system-defined limits rather than arbitrary constants
- ④ Do not modify input parameter values unless it makes sense to do so

■ Argument arrays

- an argument array
 - an array of pointers to strings
 - the shell parses the command line into tokens and passes the result to the program
- parameter에 들어가는 인자가 한 index에 하나씩 들어가 있음.



- ⑤ Avoid static variables or dynamic memory allocation if automatic allocation will do just as well ⇒ 일반적으로 automatic variable 활용

■ Thread-safe `strtok()` → line 단위로 tokenizing

- Problem of current `strtok()` → `char *strtok(char *str, const char *delim);`
 - 내부 static variable을 사용함. *static variable*
 - 같은 프로그램 내에서 다르게 parse된 string의 `strtok` → 각각의 string 간섭이 일어남. *next token*

ex) "a^b^c^d^e^f^g^h^i^j^k^l^m^" → "a^b^c^d^e^f^g^h^i^j^k^l^m^"

• `char* strtok_r(char* s, const char* sep, char **lasts)`

- POSIX : Thread-safe function 제공
- `lasts` : `strtok_r`이 다음 parse를 시작할 주소를 저장하는데 사용할 포인터 위치 (*user가 제공*)

- Use of static variables

- function call과 내부 안정적인 정보를 저장하는데 유용함
 - △ but multiple thread 에서는 관리가 필요
- static variable and function 은 외부에서 접근 가능하지 못 하도록 개런티가 있는 내부 linkage를 가짐.
- static variable : 어떤 접근하는 함수든 외부와 접근할 수 있다.

- Analyze all the calls to the malloc family

Process environment

① environment list

- an array of pointers to strings of the form name = value
 - name : an environment variable
 - value : a string associated with the name
- extern char **environ(ISO C) *array of string*
 - pointers to the process environment list when the process begins executing
 - return array of string

② Environment variables(환경변수)


- provide a mechanism for using system-specific or user-specific information in setting defaults within a program
- ex. path, home, directory, etc.
- char* getenv(const char* name)
 - to determine whether a specific variable has a value
 - return NULL ~~if~~ the variable does not have a value

POSIX environment variables

variable	Meaning
COLUMNS	Preferred width in columns for terminal
HOME	User's home directory
LANG	Locale when not specified by LC_ALL or LC_*
LC_ALL	Overriding name of locale
LC_COLLATE	Name of locale for collating information
LC_CTYPE	Name of locale for character classification
LC_MESSAGES	Name of locale for negative or affirmative responses
LC_MONETARY	Name of locale for monetary editing
LC_NUMERIC	Name of locale for numeric editing
LC_TIME	Name of locale for date/time information
LINES	Preferred number of lines on a page or vertical screen
LOGNAME	Login name associated with a process
PATH	Path prefixes for finding executables
PWD	Absolute pathname of the current working directory
SHELL	Pathname of the user's preferred command interpreter
TERM	Terminal type for output
TMPDIR	Pathname of directory for temporary files
TZ	Time zone information

Process termination

① Normal termination

- Return from main
- Implicit return from main
- Call to exit, _Exit, _exit functions
 - take a parameter of terminating status (0 if successful)
- **exit()**
 - calls user-defined exit handlers that were registered by atexit() in the reverse
- **int atexit(void (*func)(void))** → return, input type 모두 void인 함수만 다룰 수 있음. 

② Abnormal termination

- abort function(프로그램 중단)
- signal that causes termination(termination 유발하는 signal)
- may produce a core dump.