Unix System Programming
Process Environment

Chandravva Hebbi
Department of Computer Science and Engineering
chandravvahebbi@pes.edu

Unix System Programming

Process Environment

Chandravva Hebbi

Department of Computer Science and Engineering

UNIX SYSTEM PROGRAMMING

Topics to be Covered

- Command line arguments
- Environment List
- Environment Variables
- setjmp and longjmp Functions
- getrlimit and setrlimit Functions

UNIX SYSTEM PROGRAMMING

Environment Variables

- When a program is executed, the process that does the exec can pass command-line arguments to the new program.

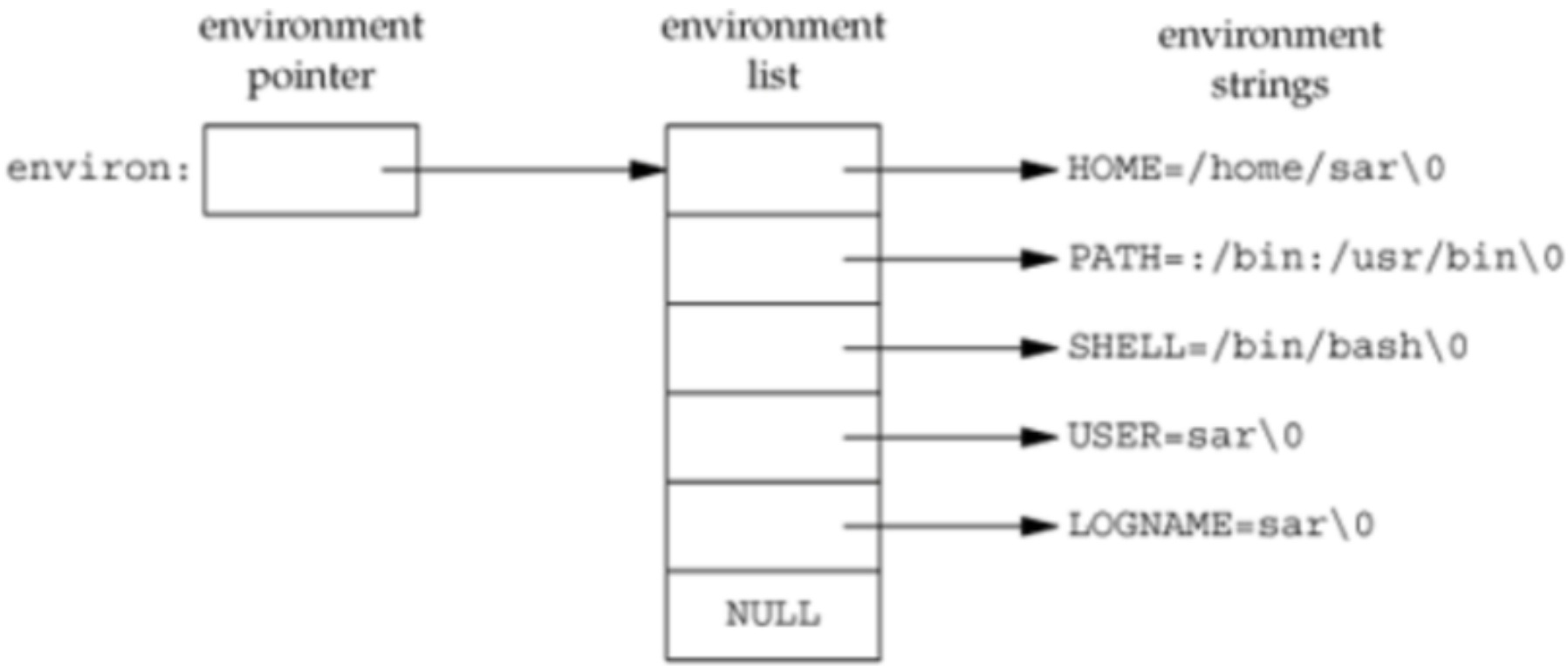
- Program Example

```
int main(int argc, char *argv[])
{
int i;
for (i = 0; i < argc; i++)
printf("argv[%d]: %s\n", i, argv[i]);
exit(0);
}
```

Environment List

- Each program is also passed an *environment list.*
- The environment list is an array of character pointers.
- Each pointer containing the address of a null-terminated C string.
- The address of the array of pointers is contained in the global variable environ.
- extern char **environ;

Environment List



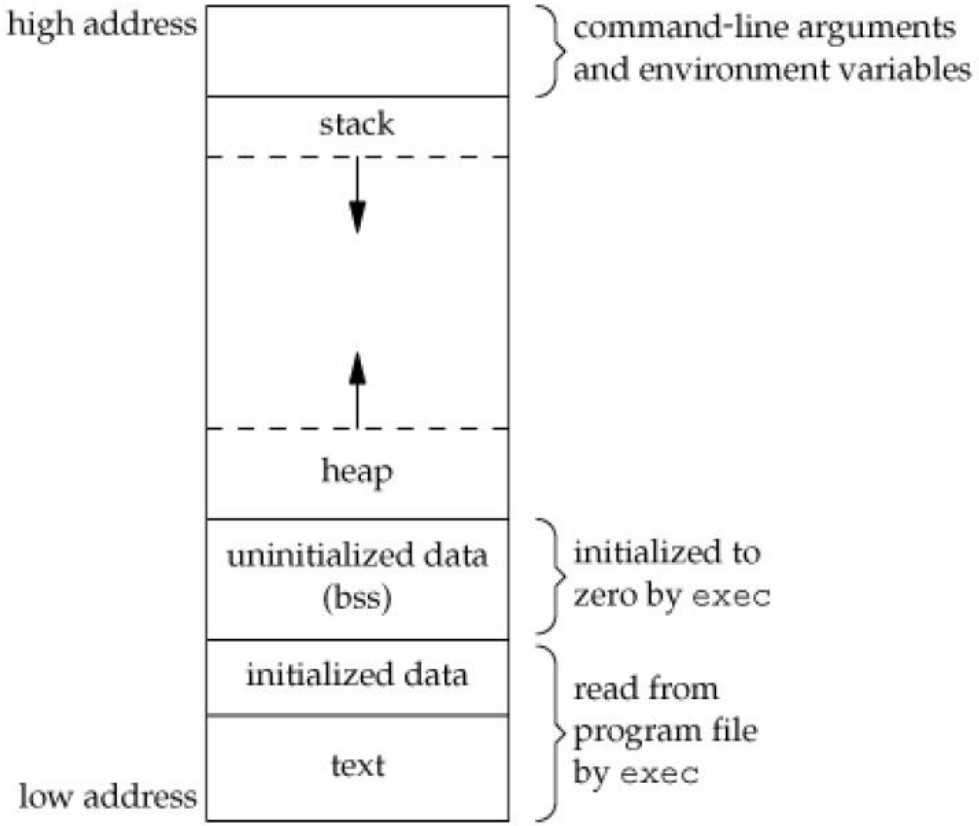
The environment consists of,
name=value

Environment List

Third argument to the main function that is the address of the environment list

- int main(int *argc*, char **argv*[], char **envp*[]);
- Access to specific environment variables is normally through the getenv and putenv functions.

Memory Layout of a C Program



Typical Memory Layout of a C program

Memory Layout of a C Program

Text segment:

- The machine instructions that the CPU executes.
- The text segment is sharable so that only a single copy needs to be in memory for frequently executed programs, such as text editors, the C compiler, the shells, and so on.
- The text segment is often readonly, to prevent a program from accidentally modifying its instructions.

Initialized data segment:

- Simply called as the data segment, containing variables that are specifically initialized in the program.

For example int n=10;

appearing outside any function causes this variable to be stored in the initialized data segment with its initial value.

Memory Layout of a C Program

Uninitialized data segment:

- Often called the "bss" segment, "block started by symbol".
- Data in this segment is initialized by the kernel to arithmetic 0 or null pointers before the program starts executing.

The C declaration

```
long sum[1000];
```

If, appearing outside any function causes this variable will be stored in the uninitialized data segment.

Memory Layout of a C Program

Stack

- Automatic variables are stored, along with information that is saved each time a function is called.
- Each time a function is called, the address of where to return to and certain information about the caller's environment, such as some of the machine registers, are saved on the stack.
- The newly called function then allocates room on the stack for its automatic and temporary variables.

Heap

- Dynamic memory allocation usually takes place from Heap.
- The heap has been located between the uninitialized data and the stack.

Memory Layout of a C Program

Several more segment types exist in an a.out, containing the symbol table, debugging information, linkage tables for dynamic shared libraries, and the like.

The size(1) command reports the sizes (in bytes) of the text, data, and bss segments.

text	data	bss	dec	hex	filename
2019	632	8	2659	a63	a.out

Environment Variables

The environment strings are usually of the form

name=value

- The shells use numerous environment variables like HOME, PATH
- ISO C defines a function that we can use to fetch values from the environment.

```
#include <stdlib.h>
char *getenv(const char *name);
```

Returns: pointer to *value* associated with *name*, `NULL` if not found

Environment Variables

The environment strings are usually of the form

name=value

- The shells use numerous environment variables like HOME, PATH
- ISO C defines a function that we can use to fetch values from the environment.

```
#include <stdlib.h>
char *getenv(const char *name);
```

Returns: pointer to value associated with name, NULL if not found

getenv is used to fetch a specific value from the environment, instead of accessing **environ** directly.

Environment Variables

```
#include <stdlib.h>

int putenv(char *str);

int setenv(const char *name, const char *value,

int rewrite);

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

All return: 0 if OK, nonzero on error


```
#include <stdlib.h>

int putenv(char *str);

int setenv(const char *name, const char *value,

int rewrite);

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

All return: 0 if OK, nonzero on error

setjmp and longjmp Functions

```
#include <setjmp.h>
int setjmp(jmp_buf env);
  Returns: 0 if called directly, nonzero if returning from a call to longjmp

void longjmp(jmp_buf env, int val);
```

Goto label

Label:

goto label1

Fun()

{ label1: }

setjmp and longjmp Functions

```
#include<setjmp.h>
jmp_buf buf;
void func()
{     printf("Welcome to GeeksforGeeks\n");      // Jump to the
point setup by setjmp
      longjmp(buf, 1);
      printf("Geek2\n");
}
int main()
{ // Setup jump position using buf and return 0
      if (setjmp(buf))
            printf("Geek3\n");
      else
      {           printf("Geek4\n");
            func();
      }
      return 0;
}
```

getrlimit and setrlimit Functions

Every process has a set of resource limits, some of which can be queried and changed by the getrlimit and setrlimit functions.

```
#include <sys/resource.h>
int getrlimit(int resource, struct rlimit *rlptr);
int setrlimit(int resource, const struct rlimit *rlptr);

struct rlimit {
rlim_t rlim_cur; /* soft limit: current limit */
rlim_t rlim_max; /* hard limit: maximum value for rlim_cur */
};
```

getrlimit and setrlimit Functions

Rules for changing resource limits

1. A process can change its soft limit to a value less than or equal to its hard limit.

2. A process can lower its hard limit to a value greater than or equal to its soft limit. This lowering of the hard limit is irreversible for normal users.

3. Only a superuser process can raise a hard limit.

THANK YOU

Chandravva Hebbi
Department of Computer Science and Engineering

chandravvahebbi@pes.edu