Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Software Systems

Lectures Week 9

Introduction to Systems Programming 1

(Systems, Concurrency, Inter process communication)

Prof. Joseph Vybihal

Computer Science

McGill University

Part A

Introduction to Systems Programming

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

About Systems Programming

Software that interacts with the computer and not only with human users

There are three basic ways to interact with your system:

- The Shell
- The OS through libraries
- Directly with the devices connected to your machine

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

The Shell

There are three basic ways to interact with the shell:

- The command-line arguments (this lecture)
- Executing shell commands (later)
- The shell memory (later)

Unix
Bash
C
GNU
Systems

C Shell Arguments

Bash-prompt $ ./a.out 15 Bob 4.2 X

```
int main(int argc, char *argv[]) {
    :
    :
}
```

argc ← counts the number of arguments
argv ← an array of strings, each cell is one argument

In this example:

argc = 4
argv = {“a.out”, “15”, “Bob”, “4.2”, “X”}

C Shell Arguments

Bash-prompt $ ./a.out 15 Bob 4.2 X

```
int main(int argc, char *argv[]) {
    int a; char name[30]; float b; char c;

    // Assume I am expecting 4 arguments
    if (argc != 4) exit(1);

    a = atoi(argv[1]);
    strcpy(name, argv[2]);        // a string can be copied (or referenced)
    b = atof(argv[3]);
    c = *argv[4];                 // copies the single character (but really…)
}
```

Example

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]) {
  char c; FILE *p, *q;

  if (argc != 2) exit(1);

  p = fopen(argv[1],”rt”);
  q = fopen(argv[2],”wt”);
  if (p==NULL || q==NULL) exit(2);

  c = fgetc(p);
  while(!feof(p)) {
          fputc(c,q);
          c = fgetc(p);
  }

  fclose(p);
  fclose(q);
  return 0;
}
```

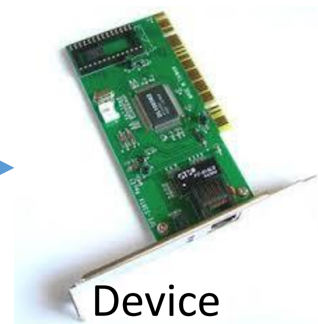
```
Bash-prompt $ vi copy.c
Bash-prompt $ gcc –o copy copy.c
Bash-prompt $ ./copy file1.txt file1.bak
```

Libraries as system interfaces

Every machine connected to your computer passes through the same pipeline:

Driver (assembler)

Knows how to interface with the specific card



Device interface card

Random device

OS

Manages all drivers

Language library

Facilitates interface by hiding details

Your program

Example

#include<time.h>

- Connects to the system clock
  - time_t seconds;   // struct stores time since Jan 1 1970
  - struct tm   // parsed date & time structure
    - tm_year, tm_ mon, tm_ mday, tm_ hour, tm_ min, tm_ sec, tm_ isdst
  - seconds = time(NULL);// local time since Jan 1 1970
    - printf(“Hours since 1970: %d”, seconds/3600);
  - float x = difftime(secondsX, secondsY);
  - struct tm *t = localtime(&seconds);
  - char *p = asctime(t);
  - seconds = mktime(t);

The time.h library

struct tm

int	tm_sec	seconds [0,59]
int	tm_min	minutes [0,59]
int	tm_hour	hour [0,23]
int	tm_mday	day of month [1,31]
int	tm_mon	month of year [0,11]
int	tm_year	years since 1900
int	tm_wday	day of week [0,6] (Sunday = 0)
int	tm_yday	day of year [0,365]
int	tm_isdst	daylight savings flag

time_t

An unsigned long integer or unsigned long double number (depends on implementation) that measures the number of milliseconds from a fixed point in time to the present as an offset (or distance measurement).

Jan 1 1970 is the earliest date/time that can be represented. Jan 1 1970 = 0.

The time.h library

```
char *        asctime(const struct tm *);
char *        asctime_r(const struct tm *, char *);
clock_t       clock(void);
int           clock_getres(clockid_t, struct timespec *);
int           clock_gettime(clockid_t, struct timespec *);
int           clock_settime(clockid_t, const struct timespec *);
char *        ctime(const time_t *);
char *        ctime_r(const time_t *, char *);
double        difftime(time_t, time_t);
struct tm *getdate(const char *);
struct tm *gmtime(const time_t *);
struct tm *gmtime_r(const time_t *, struct tm *);
struct tm *localtime(const time_t *);
struct tm *localtime_r(const time_t *, struct tm *);
time_t        mktime(struct tm *);
int           nanosleep(const struct timespec *, struct timespec *);
size_t        strftime(char *, size_t, const char *, const struct tm *);
char *        strptime(const char *, const char *, struct tm *);
time_t        time(time_t *);
int           timer_create(clockid_t, struct sigevent *, timer_t *);
int           timer_delete(timer_t);
int           timer_gettime(timer_t, struct itimerspec *);
int           timer_getoverrun(timer_t);
int           timer_settime(timer_t, int, const struct itimerspec *, struct itimerspec *);
void          tzset(void);
```

Example

Using time.h as a profiler.

```
#include <stdio.h>
#include <time.h>
 int main()
 {
   time_t begin,end;
   long i;

   begin= time(NULL);
   for(i = 0; i < 150000000; i++);
   end = time(NULL);

   printf("for loop used %f seconds to complete the execution\n", difftime(end, begin));

   return 0;
}
```

Unix
Bash
C
GNU
Systems

Directly with devices connected to the computer

As a system's language, C permits direct access to devices.

- A device card has an address
  - void *p = 145; // assume we know the card's address is 145
- We can communicate with the device
  - *p = 'A'; // if p points to printer then printer prints 'A'
  - int x = *p; // if p points to printer status then x = status
- Devices speak in binary…

Binary

Counting in decimal and binary:

0	00000000
1	00000001
2	00000010
3	00000011
4	00000100
5	00000101

In Decimal: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
In Binary : 0, 1

In decimal: 9 + 1 = 10 (we reuse digits, = 10)

In binary: 1 + 1 = 10 (we reuse digits, = 2)

Binary is used in many ways:

- The char is ASCII which is coded binary
- Numbers are stored as binary
- Binary 00101 can be thought of as three false values and two true values

Manipulating binary in C

Operators:

- `&` Binary and
- `|` Binary or
- `~` Binary complement
- `>>` Binary shift right
- `<<` Binary shift left

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Manipulating binary in C

Meaning:

&	1011 & 0110 → 0010
|	1011 | 0110 → 1111
~	~1011 → 0100
>>	1011>>3 → 0001
<<	1011<<2 → 1100

AND:

A	B	AND	R
1	1		1
1	0		0
0	1		0
0	0		0

OR:

A	B	OR	R
1	1		1
1	0		1
0	1		1
0	0		0

Complement:
Means oposite

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Manipulating binary in C

Usage:

```
int a = 11; // Binary 1011
int b = 6;  // Binary 0110
int r;
r = a & b;  //     1011 & 0110 → 0010
r = a | b;  //     1011 |  0110 → 1111
r = ~a;     //     ~1011 → 0100
r = a>>3;   //     1011>>3 → 0001
r = a<<2;   //     1011<<2 → 1100
```

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Example

Masking:

- When we want to change a single bit, or
- When we want to find out about a bit.

```
int b = 6;   // Binary 0110

b = b | 0001;      // "set" the last bit to 1
b = b &1110;       // "set" the last bit to 0

if (b & 0001)      // =0 if last bit was 0, else =1
```

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Part B

Concurrent Programs

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

What is a concurrent program?

It is when two programs are running at the same time on the same machine.

It does not mean that they are cooperating, but often they are.

Ampersand and semi-colon

- Sequential command-line execution
  - Example:
    - ls; cp file1.c /backup; cat file1.c
  - In the above example the ls command is executed first. After the ls command is finished the cp command executes. After cp is finished then the cat command runs last.
- Concurrent command-line execution
  - Example:
    - ls & cp file1.c /backup & cat file1.c
  - In this example all three commands execute concurrently.
  - They do not launch at the same time – if they run long enough they will all use the CPU at the same time.
  - Any concurrent output is displayed to the screen at the same time… mixed together.

Using the Shell for Concurrency

- ;
  - Sequential execution of commands
  - Example: who; grep 'Jack' eg.txt; ls > out.txt
- &
  - Parallel execution of commands
  - Example: who & whoami & ls

What would the output look like when invoked on the command line?

Related Commands

- PS
  - See all the currently running programs (processes). Notice that each process as an ID number called the PID.
  - Syntax to see your own: ps
  - Syntax to see everyone: ps -e
- KILL
  - Terminate an executing program (process)
  - Syntax standard : kill PID
  - Syntax emergency: kill -9 PID

Active Processes

- The `ps` command is an ideal solution for troubleshooting problem processes.
- Although the command options have a tendency to change from one OS to another, here are some of the common options.
  - -a : all processes, all users
  - -e : environment/everything
  - -g : process group leaders as well
  - -l : long format
  - -u : user oriented report
  - -x : even processes not executed from terminals
  - -f : full listing

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Demo

- Semi-colon
- Ampersand
- PS
- Kill

Example

Bash-prompt $ vi producer.c

Bash-prompt $ vi consumer.c

Bash-prompt $ gcc –o producer producer.c

Bash-prompt $ gcc –o consumer consumer.c

Bash-prompt $ ./producer & ./consumer

These two programs runs concurrently without using fork().

These use a text file to coordinate and share the data (as we saw last class).

Unix
Bash
C
GNU
Systems

Example

```
// PRODUCER.C
int main() {
  char c = ‘ ‘;
  FILE *p;

  while (c != ‘x’) {
     c = getchar();

     while ((p=fopen(“shared.txt”,”at”) == NULL) ;
     fprintf(p,”%c\n”, c);
     fclose(p);
  }

  return 0;
}
```

Unix
Bash
C
GNU
Systems

```
// CONSUMER.C
int main() {
   char c; int pos = 0;
   FILE *q;

   do {
      c = removeCharacter(pos);
      printf(“%c”, c);
      pos++;
   } while (c != ‘x’);
}
char removeCharacter(int pos) {
   char c;
   FILE *q;
   int x = 0;

   while ((q=fopen(“shared.txt”, “rt”) == NULL)  ;

   c=fgetc(q);
   while(!feof(q) && x < pos) {
            pos++;
            c=fgetc(q);
   }
   fclose(q);
   return c;
}
```

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Accessing Shell Memory

Remember:

Bash-prompt $ set

The ‘set’ command displays all the variables defined within the shell memory.

C can access these variables:

```
#include <stdlib.h>
char *data = getenv(“VARIABLE_NAME”);
```

```
printf("PATH : %s\n", getenv("PATH"));
printf("HOME : %s\n", getenv("HOME"));
printf("ROOT : %s\n", getenv("ROOT"));
```

The getenv() returns NULL if it failed to find the variable.

```
int n = atoi(getenv(“CONTENT_LENGTH”));
```

```
int setenv (const char *name, const char *value, int replace)
```

```
int success = setenv(“VAR_NAME”, “VALUE”, 1);
```

Replace = 1 for true, 0 for false
Success = 0 for true, -1 for false

Example

Bash-prompt $ ./prog1; ./prog2

Notice that prog1 runs to completion first before prog2 starts.

In prog1:

- setenv(“x”, “yes”, 1);

In prog2:

- char *p = getenv(“x”);

Notice that a preceding program can communicate with a following program.

Example

Bash-prompt $ set x="yes"

Bash-prompt $ ./prog2

User sets the variable at the command prompt

In prog2:

- char *p = getenv("x");

Notice that the user can change the "environment" causing prog2 to behave differently.

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Example

Bash-prompt $ ./prog3 & ./prog4

Two concurrent programs can coordinate using the shell memory

Assume a shell variable TURN is “3” or “4”.

In prog3:

```
char *p;
while (1) {
   do { p = getenv(“TURN”); }
   while (strcmp(p, “3”) != 0) ; // wait
   … do something since it is 3 now …
   setenv(“TURN”, “4”, 1);
}
```

In prog4:

```
char *q;
while (1) {
   do { q = getenv(“TURN”); }
   while (strcmp(q, “4”) != 0) ; // wait
   … do something since it is 4 now …
   setenv(“TURN”, “3”, 1);
}
```

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Question

Suggest a way we could use shell memory to do the producer/consumer problem. Use the ampersand to launch the two programs.

In this case we would like to simply produce one value and consume that one value, and repeat. The user enters values from the keyboard until they write the word DONE, which then terminates both programs.

Part C

Inter-process Communication

Readings: http://www.csl.mtu.edu/cs4411.ck/www/NOTES/process/fork/create.html

What is a process?

Program

- A program is understood to mean a file on disk containing a runnable algorithm (compiled or interpret-able)

Process

- A copy of the program but in RAM (the computer's live memory) and it is currently being executed

Unix
Bash
C
GNU
Systems

Program & Process

PROGRAM

- Loader
- Static Data
- Algorithms
- Libraries
- DLL requests

Tells OS what resources program needs to run

DLL libraries

Are attached to OS and shared with all processes

PROCESS

- Run-time Heap
- Run-time Stack
- Static Data
- Algorithms
- Libraries
- DLL requests

malloc
calloc
new

Local variables, recursion

COMP 206 – Joseph Vybihal
Software Systems

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Three types of processes

- Shell processes
  - Is a process that runs within its own shell. Launching a process of this form will pause the current shell by launching a new shell and then using the command-line of the new shell to launch a program or execute a shell command.
- Cloned process
  - The currently executing process can clone itself. After the clone operation there are two identical processes running in the same shell concurrently.
- New process
  - The currently executing process can launch a new program within the current shell. Both processes run concurrently.

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Three types of processes

- Shell processes
  - #include<stdlib.h>
  - int system(char *command_line_command);
  - Returns -1 on error, or return status number
- Cloned process
  - #include<unistd.h>
  - int fork();
  - Returns the process ID number
- New process
  - Not covered…

Unix
Bash
C
GNU
Systems

The system() function

Bash-prompt $ ./a.out

start
a.out

```c
int main() {
  char *c="./a2.out 5";
  int result;

  result=system(c);

  return result;
}
```

a2.out
starts

a.out
sleeps

a2.out
kill -9

a.out
ends

```c
int main(int argc, char *argv[]) {
  int x;

  for(x=0;x<10;x++)
    printf("%d\n",x);

  return 0;
}
```

Programs can use system() as often as needed.
A process invoked by system() can also use system(), recursive definition.

Notice how data is passed through command-line arguments & return statement.

COMP 206 – Joseph Vybihal
Software Systems

The fork() function

```
#include <stdlib.h> /* needed to define exit() */
#include <unistd.h>/* needed for fork() and getpid() */
#include <stdio.h>  /* needed for printf() */

 int  main(int argc, char *argv[]) {
        int pid;    /* process ID */

        switch (pid = fork()) {
        case 0:             /* fork returns 0 to the child */
                printf("I am the child process: pid=%d\n", getpid());
                break;

        default:    /* fork returns the child's pid to the parent */
                printf("I am the parent process: pid=%d, child pid=%d\n", getpid(), pid);
                break;

        case -1:    /* something went wrong */
                perror("fork"); // send string to STDERR
                exit(1);
        }
        exit(0);
}
```

Programs can use fork() as often as needed.
A process invoked by fork() can also use fork().

Unix
Bash
C
GNU
Systems

The fork() function

Bash-prompt $ ./a.out

start
a.out

x=
pid

```
int main() {
   :A:
   x = fork();
   :B:
   if (x == 0) {
      :C:
   } else {
      :D:
   }
   :E:
   return 0;
}
```

a.out
ends

clone
starts
at fork!

```
int main() {
   :A:
   x = fork();
   :B:
   if (x == 0) {
      :C:
   } else {
      :D:
   }
   :E:
   return 0;
}
```

x = 0

a2.out
kill -9

a.out → :A: :B: :D: :E:
clone → :B: :C: :E:
The clone receives a copy of all the variables (it is not shared).

COMP 206 – Joseph Vybihal
Software Systems

The producer & consumer problem

The producer

- A program that generate data saving that data in a data structure or in a file.

The consumer

- A program that reads data from a data structure or a file performing some kind of computation.

Concurrency

- For some reason the producer and consumer need to run independently but cooperatively.
  - Maybe the computation is slow but the data comes in quickly. The intermediate data structure or file is used as a temporary cache.

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Example

```
#include <stdlib.h> #include <unistd.h> #include <stdio.h>

int  main() {
        int pid = fork();
        if (pid == -1) exit(1);
        if (pid == 0) { producer();  wait(); }  // wait for consumer to end
        if (pid != 0) { consumer(); wait(); } // wait for child to end
}

void producer() {
        :
        :
}

void consumer() {
        :
        :
}
```

Not an efficient example, but easy to understand.

Notice the use of functions to make the code easier to read.

Unix
Bash
C
GNU
Systems

Example

```
void producer() {
   char c = ‘ ‘;
   FILE *p;

   while (c != ‘x’) {                        // program stops at the input of ‘x’
      c = getchar();

      while ((p=fopen(“shared.txt”,”at”)) == NULL) ; // notice semi-colon, we wait…
      fprintf(p,”%c\n”, c);
      fclose(p);                             // give the other process a change to open the file
   }
}

void consumer() {
   char c; int pos = 0;
   FILE *q;

   do {
      c = removeCharacter(pos);   // we encapsulate further here for understandability
      printf(“%c”, c);
      pos++;                      // pos tracks the next data, pos is independent of producer
   } while (c != ‘x’);            // program stops at input of ‘x’
}
```

Data sharing is handled by a text file cache.

COMP 206 – Joseph Vybihal
Software Systems

Example

```
char removeCharacter(int pos) {
   char c;
   FILE *q;
   int x = 0;

   while ((q=fopen("shared.txt", "rt") == NULL)  ;  // busy wait

   c=fgetc(q);
   while(!feof(q) && x < pos) {
              x++;                                   // move to the correct position in file
              c=fgetc(q);
   }

   fclose(q);

   return c;                                         // return the character
}
```

How can we do this with fseek?

Notice that the two programs run independently from each other.
The speed by which they processed the file was different and did not matter.
However the current implementation assumes that producer is faster than consumer.

Question

The previous example assumed that producer was faster than consumer.

How could we modify consumer to handle the case of being sometimes faster than producer?

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Using shared memory

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

void *shmat(int shmid, const void *shmaddr, int shmflg);
```

- Attach shared memory to program

```
int shmdt(const void *shmaddr);
```

- Detach shared memory from program

```
int shm_id = shmget(
        key_t    k,        /* the key for the segment      */
        int      size,     /* the size of the segment      */
        int      flag);    /* create/use flag              */
```

- Creates the shared memory space

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Example

```
#include  <sys/types.h>
#include  <sys/ipc.h>
#include  <sys/shm.h>
#include  <stdio.h>
    .....
 int       shm_id;        /* shared memory ID      */
 struct MEM { int a, b, c, status; } *p;   // status=0 empty, 1 filled
    .....
shm_id = shmget(IPC_PRIVATE, sizeof(struct MEM), IPC_CREAT | 0666);
if (shm_id < 0) exit(1);

/* now the shared memory ID is stored in shm_id */

 p = (int *) shmat(shm_id, NULL, 0);
 if (p == -1) exit(1);

 p->a = 5;
 p->status = 1;
 while(p->status == 1) sleep(1);  // wait for child, child will change status to 0
 exit(0);
```

IPC_CREAT | 0666 for a server (*i.e.*, creating and granting read and write access to the server) - Octal

0666 for any client (*i.e.*, granting read and write access to the client) - Octal

If a client wants to use a shared memory created with **IPC_PRIVATE**, it must be a child process

Child program is similar.
Examples: HERE and HERE.