II. *A Crash Course in C*

Jalal Kawash

rchitecture

Section 1 Objectives

At the end of this section you will

1. Learn how to write programs in C
2. Use C pointers and athematic
3. Exploit I/O files
4. Create processes and threads
5. Mix C and Assembly

rchitecture

Outline

1. Basics
2. Functions
3. Branching and looping
4. Arrays
5. Pointers
6. Structures and memory allocation
7. Files
8. Processes
9. Threads
10. Mixing C and Assembly

Basics

Example C Program

ex.c

```c
#include <stdio.h>
void main() {
    int n;
    char fname[20], lname[20];
    float cost;
    printf("Enter your name:");
    scanf("%s %s",fname,lname);
    printf("How many items are there?");
    scanf("%d",&n);
    printf("What is the unit cost?");
    scanf("%f",&cost);
    printf("\nHey %s %s, \nThere are %d items at $%4.2f\n", fname, lname, n, cost);
}
```

Basic Data Types

Type	Bytes	Use
int	4	
char	1	
float	4	
double	8	
short	2	short int
long	8	long int
unsigned	4	unsigned int

Conversion and Casting

- C can do type conversions
- In general, a smaller-size type can be converted to a larger-size type
  - E.g. float = int

- Explicit conversion is done as follows:
  - (datatype) expression
  - avg = (float) sum/total

Basic Operators

- Addition
- Subtraction
- Multiplication
- Division
- Modulus division

Functions

Functions

```
int isEven(long int n) {
    return !(n % 2);
}

int isOdd(long int n) {
    return (n % 2);
}

int max(long int n1, long int n2) {

    if (n1 > n2)
        return n1;

    return n2;
}
```

Value Parameters

- Do not change outside the function

valueParam.c

```c
#include <stdio.h>

void func(int n, int m) {
    n = 8;
    m = 9;
}

int main() {
    int n = 0, m = 0;
    func(n,m);
    printf("%d, %d", n, m);
}
```

Output: 0, 0

Reference Parameters

- Change outside the function

refParam.c

```
1  #include <stdio.h>
2
3  void func(int *n, int *m) {
4      *n = 8;
5      *m = 9;
6  }
7
8  int main() {
9      int n = 0, m = 0;
10     func(&n,&m);
11     printf("%d, %d", n, m);
12 }
```

`*n` is a pointer = variable holding the address on `n`

`&n` = address on `n`

Output: 8, 9

Command-line Arguments

- Mechanism to collect user input from the command line:

  `prog <list of arguments>`

- The `main` method can have two arguments:
  - `argc` = argument count
  - `argv` = argument vector

```
int main(argc,argv)
    int argc;
    int *argv[];
{
…
}
```

Command-line Arguments

- `argc` is computed by the compiler
- `argv[0]` = program name

cmdLnArgs.c

```c
#include <stdio.h>

int main(argc, argv)
    int argc;
    char *argv[];
{
    int n = 0;
    if (argc == 1) return 1;
    while (argv[1][n++] != '\0');
    printf("\"%s\" has %d letters",argv[1],n);
}
```

```c
#include <stdio.h>

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

Branching and Looping

Comparison and Logical Operators

- Comparison: ==, !=, <=, >=
- Logical:
  - AND: &&
  - OR: ||
  - NOT: !

If Statement

grades.c

```c
#include <stdio.h>

int main () {
    int m;
    printf("Enter your mark:");
    scanf("%d",&m);
    if ( m > 100 || m < 0) { printf("Incorrect mark."); return 1;}
    else if (m >= 95) printf("A+");
    else if (m >= 90) printf("A");
    else if (m >= 85) printf("A-");
    else if (m >= 80) printf("B+");
    else if (m >= 75) printf("B");
    else if (m >= 70) printf("B-");
    else if (m >= 65) printf("C+");
    else if (m >= 60) printf("C");
    else if (m >= 55) printf("C-");
    else if (m >= 50) printf("D+");
    else printf("F");

    return 0;
}
```

Switch Statement

```
switch (n) {
    case 9: printf("Nine"); break;
    case 8: printf("Eight"); break;
    case 7: printf("Seven"); break;
    case 6: printf("Six"); break;
    case 5: printf("Five"); break;
    case 4: printf("Four"); break;
    case 3: printf("Three"); break;
    case 2: printf("Two"); break;
    case 1: printf("One"); break;
    case 0: printf("Zero"); break;
}
```

Bitwise Operators

Bitwise AND	&
Bitwise OR	|
Bitwise XOR	^
Left Shift	<<
Right Shift	>>
One's complement	~

Examples

- `value = eFlag & mask`
- `value = eFlag >> n`
- `value = eFlag << 3`
- `value ~= eFlag`
- `value |= eFlag`
- `value = (~0 ^ eFlag) << 5`

While Loop

```
while (condition) {
body
}
```

numToLet.c

```c
#include <stdio.h>
void main() {
    long int n, r, sum=0;

    void numToLetter(long int n){
        switch (n) {
            case 9: printf("Nine"); break;
            case 8: printf("Eight"); break;
            case 7: printf("Seven"); break;
            case 6: printf("Six"); break;
            case 5: printf("Five"); break;
            case 4: printf("Four"); break;
            case 3: printf("Three"); break;
            case 2: printf("Two"); break;
            case 1: printf("One"); break;
            case 0: printf("Zero"); break;
        }
    }


    printf("Enter a non-negative number:");
    scanf("%ld", &n);
    if (n < 0) {printf("Number must be non-negative!"); return;}


    while(n > 0) {
        r = n % 10;
        sum = sum * 10 + r;
        n = n / 10;
    }

    n = sum;
    while(n > 0) {
        r = n % 10;
        numToLetter(r);
        printf(" ");
        n = n / 10;
    }
}
```

Do-While Loop

```
do {
body
} while (condition);
```

Equivalent to:

```
body
while (condition) {
body
}
```

For Loop

```
for (initialization; condition; incrementation){
Body
}
```

```
int i;
for(i = 0; i < 100; i++)
    printf("%d ",i);
```

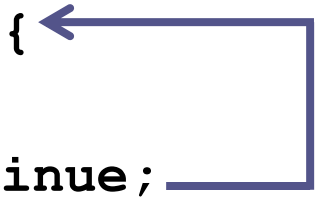
triangle.c

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

int main()
{
    int i, k, levels, space;
    printf("Enter the number of levels in the triangle:");
    scanf("%d",&levels);

    space = levels;

    for ( i = 1 ; i <= levels ; i++ )
    {
        for ( k = 1 ; k < space ; k++ )
            printf(" ");
        space--;

        for ( k = 1 ; k <= 2*i - 1 ; k++ )
            printf("*");
        printf("\n");
    }
    return 0;
}
```

continue and break

```
while (…) {
…
if(…) break;
…
}
```

```
while (…) {
…
if(…) continue;
…
}
```



Arrays

Arrays

- To declare:

```
int y[100];
int x[10][20];
```

- Static and external (to `main()`) arrays can be initialized

```
static int y[] ={1,2,3,4,5,6};
int x[2][3] = {1,2,3,4,5,6}
// row major
```

sorting.c

```c
/* Code from www.sanfoundry.com */
    #include <stdio.h>

    void main() {
        int i, j, a, n, number[30];
        printf("Enter the value of N \n");
        scanf("%d", &n);

        printf("Enter the numbers \n");
        for (i = 0; i < n; ++i)
            scanf("%d", &number[i]);

        for (i = 0; i < n; ++i) {
             for (j = i + 1; j < n; ++j) {
                if (number[i] > number[j]) {
                    a =  number[i];
                    number[i] = number[j];
                    number[j] = a;
                }
            }
        }

        printf("The numbers arranged in ascending order are given below \n");
        for (i = 0; i < n; ++i)
            printf("%d\n", number[i]);
    }
```

Pointers

Pointers

- A variable that contains an address (for another object in memory)
- The address operator & returns the address of a variable
- The indirection operator * declares a pointer variable

Pointers Example

```
int n = 100;
```

Variable `n` at some address `abc`


```
int *ptr;
```

Variable `ptr` at some address `fgh`


```
ptr = &n;
```


```
int m = *ptr
```

Variable `m` at some address `lmn`

Pointers Example

```
int m;
```

Variable `n` at some address `abc`

Variable `ptr` at some address `fgh`


```
m = ptr;
```

Variable `m` at some address `lmn`


```
m = *ptr;
```

Variable `m` at some address `lmn`

Arrays and Pointers

- The array name (`a`) is a pointer to the first element in an array (`a[0]`)

```
int a[N], *ptr;
ptr = a; // array has two names now
```

`ptr+n` or `a+n` refer to `a[n]`

arraysAndPinters.c

```c
#include <stdio.h>

void main() {
    static int a[10] = {1,2,3,4,5,6,7,8,9,10};
    int *ptr, i;

    for (i = 0; i < 10; i++)
        printf("%d ", *(a+i));

    printf("\n");
    ptr = a;
    for (i = 0; i < 10; i++)
        printf("%d ", *(ptr+i));
}
```

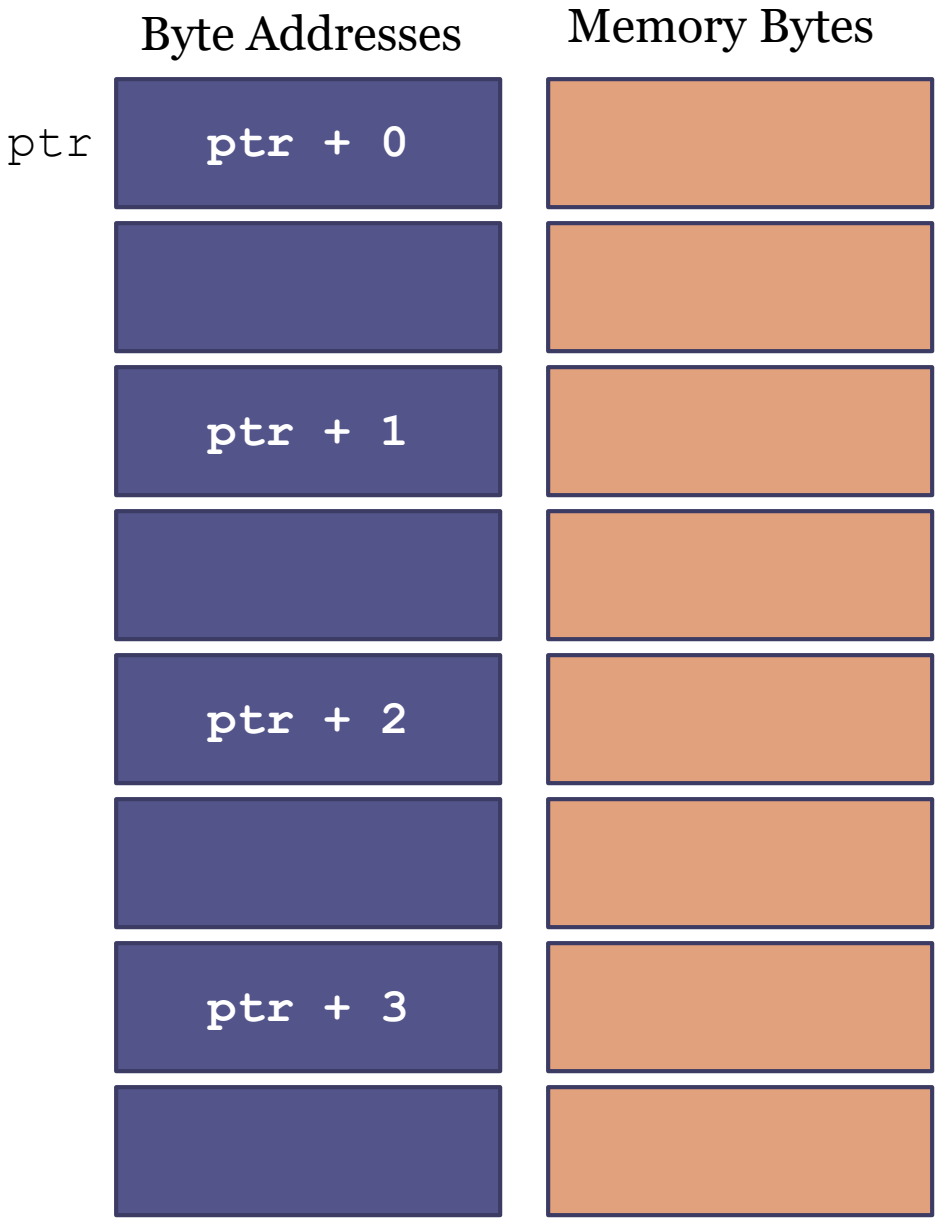
Pointer Arithmetic

```
int a[N], *ptr;
ptr = a; // array has two names now
```

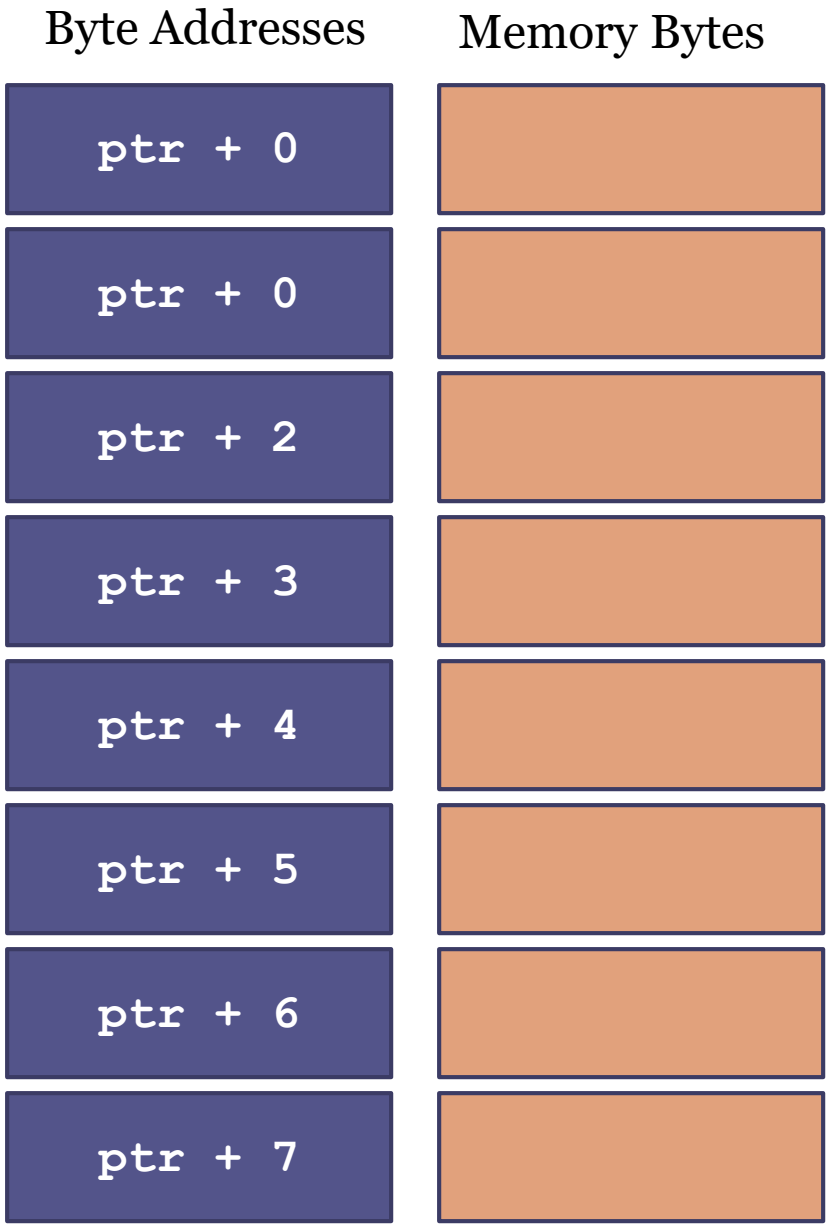
`ptr+n` or `a+n` refer to `a[n]`

- `a` and `ptr` must be the same type
- `a+n` is `n` "type size" from `a`
- Type size depend on the type of `a`
  - 2 bytes for `int`, 1 for `char`, 4 for `long`, etc …

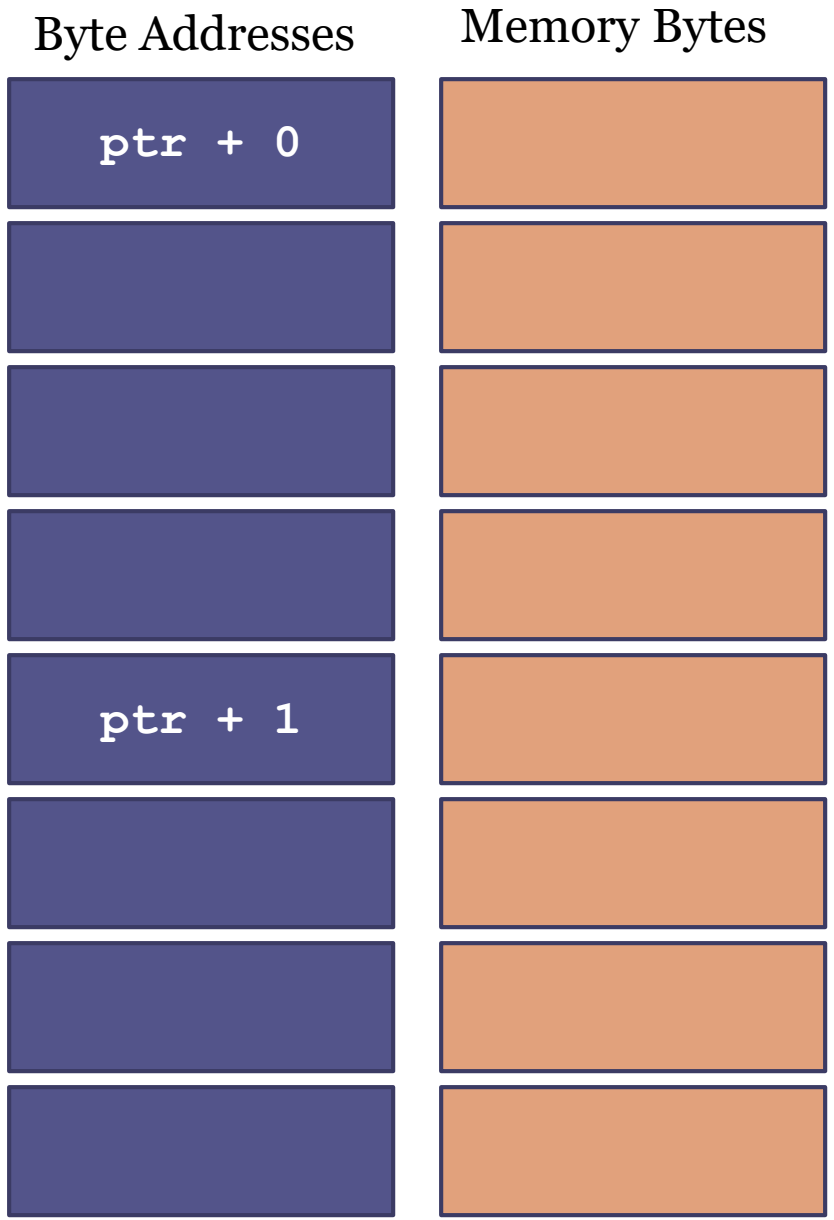
```
short *ptr;
```



```
char *ptr;
```


Byte Addresses
Memory Bytes
ptr + 0
ptr + 0
ptr + 2
ptr + 3
ptr + 4
ptr + 5
ptr + 6
ptr + 7

```
float *ptr;
```



Pointer Arithmetic

- A pointer may be displayed as an unsigned variable
- Pointers can be incremented or decremented

```
ptr++;
ptr -= 2;
```

Pointer Arithmetic Examples

```
float a[100], fptr;
fptr = &a[0];
```

- The following equalities hold
- `fptr + 5 == &a[5]`
- `++fptr + 5 == &a[1]`

Pointer Arithmetic Examples

```
float a[100], fptr;
fptr = &a[0];
fptr--;
```

- The following equalities hold
- `*fp == a[0]`
- `*(fp) == a[0]`
- `*(fp + 6) == a[6]`
- `*(fp + 6) == a + 6`

Your Turn

- What does the following declare?

```
float *a[];
```

Examples of Operations on Pointers

1. Obtaining the pointer's address

```
int n,*p1,*p2;
p1 = &n;
p2 = &p1;
```

2. A pointer can be declared as a pointer to a pointer

```
int **p2;
```

Examples of Operations on Pointers

3. Adding or subtracting

```
p1++;
```

4. Indirect operator

```
int n,m,*p1,*p2;
p1 = &n;
P2 = &m
*p1 = 20; \\ assign 20 to n
*p2 = 4 + (*p1)
\\ assign 4 + the value of n to m
```

Examples of Operations on Pointers

5. Finding the address of a pointer

```
p2 = &p1;
```

6. Finding the difference between two pointers

pointerDifference.c

```c
#include <stdio.h>
void main() {
    int n[5],i,*p1,*p2;

    p1 = n;
    for (i=0; i<5; i++) {
        p2 = &n[i];
        printf("The difference is %d\n", p2-p1);
    }
}
```

Strings

- A string is a character array terminated with NULL

```
char *name; \\ name[i] is a char
char *name[10]; \\ name[i] is a string
```

Pointers to Functions

- Declared as

```
type (*funcName)();
```

- Example: `int (*f1)();`
  - Declares a pointer to a function that returns int

- Careful: `int *f1();`
  - Declares a function returns a pointer to int

Pointers to Functions

```
void *runner(void *param)
{
  …
}

pthread_create(&tid, &attr, runner, argv[1]);
```

Structures

Structures

- Groups variables into a record

```
struct Student {
    long int id;
    char fname[20];
    char lname[20];
};

struct cpsc359 Student[50];
cpsc359[15].id = 12345678;
```

Unions

- Similar to structures in syntax, but only holds the value for one member only

```
union Answers {
    int answer1;
    float answer2;
} someAnswers;

someAnswers.answer1 = 12;
someAnswers.answer2 = 15.5;
\\ erases answer1!
```

Memory Allocation

Memory Allocation

- Declaring a pointer to a buffer will only reserve memory for the pointer, not the buffer
- Functions to allocate (reserve) memory in C:
  - malloc()
  - alloc()
  - realloc()
  - calloc()
  - free()

ptr = malloc(size)

- Allocates `size` contiguous bytes
- `ptr` is a pointer to char type
- If allocation fails, `malloc` returns `0`
- Include `#stdlib.h`

```
unsigned int size = 1024;
char *ptr;

ptr = malloc(size);
```

Allocating a buffer ot 1024 floats

```
unsigned int size = 1024;
float *ptr;

ptr = (float *)
        malloc((sizeof) float) * size);
```

`ptr = alloc(size)`

- Not available in all systems
- Similar to `malloc()`, but initializes the buffer to zeros.

ptr = realloc(ptr,newSize)

- Reallocates the old buffer of `ptr` to `newSize`
- Buffer can grow or shrink

```
unsigned int size = 1024;
char *ptr;

ptr = malloc(size);
…
ptr = realloc(ptr, size*2);
```

`ptr = calloc(n,objSize)`

- Allocates a buffer of objects
- Used to allocate memory for non-simple types
  - Such as structures
- `n` is the buffer size
- `objSize` is the size of individual object size

`free(ptr)`

- Frees the space originally allocated with pointer `ptr`

Files

Files

- Declare: `FILE *inFile, *outFile;`
- Open: `inFile = fopen("fork.c","r");`
  `outfile = fopen("fork1.c","w");`
- Read: `fscanf(inFile, "%c", ch);`
- Write: `fprintf(outFile, "%c", ch);`
- Close: `fclose(inFile);`
  `fclose(outFile);`

Files

- EOF: Delimits the end of a file

```
while (ch = fgetc(inFile) != EOF) {…}
```

- `fopen` returns NULL if unsuccessful

fileCopy.c

```c
#include <stdio.h>

int main(argc, argv)
    int argc;
    char *argv[];
{
    FILE *inp, *outp;
    char ch;

    if (argc < 3) {
        printf("Usage filecopy \"sourceFile\" \"destinationFile\"\n");
        return 1;
    }

    if ((outp = fopen(argv[2], "r")) != NULL) {
        fclose(outp);
        printf("Destination file \"%s\" already exists.\n", argv[2]);
        printf("Copy aborted.\n");
        return 1;
    }

    if ((inp = fopen(argv[1], "r")) == NULL) {
        printf("Unable to open input file \"%s\".\n", argv[1]);
        return 1;
    }

    if ((outp = fopen(argv[2], "w")) == NULL) {
        printf("Unable to open output file.\n");
        return 1;
    }

    while ((ch = fgetc(inp)) != EOF) {
        fputc (ch,outp);
    }
    fputc(EOF,outp);
    fclose(outp);
    fclose(inp);
    printf("File \"%s\" has been successfuly copied to \"%s\".\n", argv[1], argv[2]);
    return 0;

}
```

```
15      if ((outp = fopen(argv[2], "r")) != NULL) {
16          fclose(outp);
17          printf("Destination file \"%s\" already exists.\n", argv[2]);
18          printf("Copy aborted.\n");
19          return 1;
20      }
21
22      if ((inp = fopen(argv[1], "r")) == NULL) {
23          printf("Unable to open input file \"%s\".\n", argv[1]);
24          return 1;
25      }
26
27      if ((outp = fopen(argv[2], "w")) == NULL) {
28          printf("Unable to open output file.\n");
29          return 1;
30      }
```

```
32      while ((ch = fgetc(inp)) != EOF) {
33          fputc (ch,outp);
34      }
35      fputc(EOF,outp);
36      fclose(outp);
37      fclose(inp);
```

Processes

Processes

- A process is a program in execution
- A program is a passive entity
- A process is an active entity
  - Consumes resources
  - Changes states

fork.c

```c
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <sys/wait.h>
#include <stdlib.h>

int main()
{
    pid_t fr;
    fr = fork();

    if (fr < 0) {
        fprintf(stderr, "Fork Failed");
        exit(-1);
    }
    else if (fr == 0) {
        execlp("/bin/ls", "ls", NULL);
    }
    else {
        wait(NULL);
        printf("Child Completed");
        exit(0);
    }
}
```

fork.c

```
int main() {
pid_t  fr;
  fr = fork();
  if (fr < 0) {
      fprintf(stderr, "Fork Failed");
      exit(-1);
  } else if (fr == 0) {
      execlp("/bin/ls", "ls", NULL);
  }
    else { /* fr > 0 */
      wait (NULL);
      printf ("Child Complete");
      exit(0);
  }
}
```

fork.c

```
int main() {
pid_t  fr;
  /* fork another process */
  fr = fork();
  if (fr < 0) { /* error occurred */
     fprintf(stderr, "Fork Failed");
     exit(-1);
  } else …
```

fork() creates a child process.
Child process has same text section as parent process

Failed Fork()

fork.c

```
fr = fork(); // returns –ve value
if (fr < 0) { …
```

fr = -1

Parent Process

```
if (fr < 0) {
    fprintf(stderr, "Fork Failed");
    exit(-1);
}
```

fork.c

```
int main() {
pid_t  fr;
  /* fork another process */
  fr = fork();
  if (fr < 0) { /* error occurred */
     fprintf(stderr, "Fork Failed");
     exit(-1);
  }
```

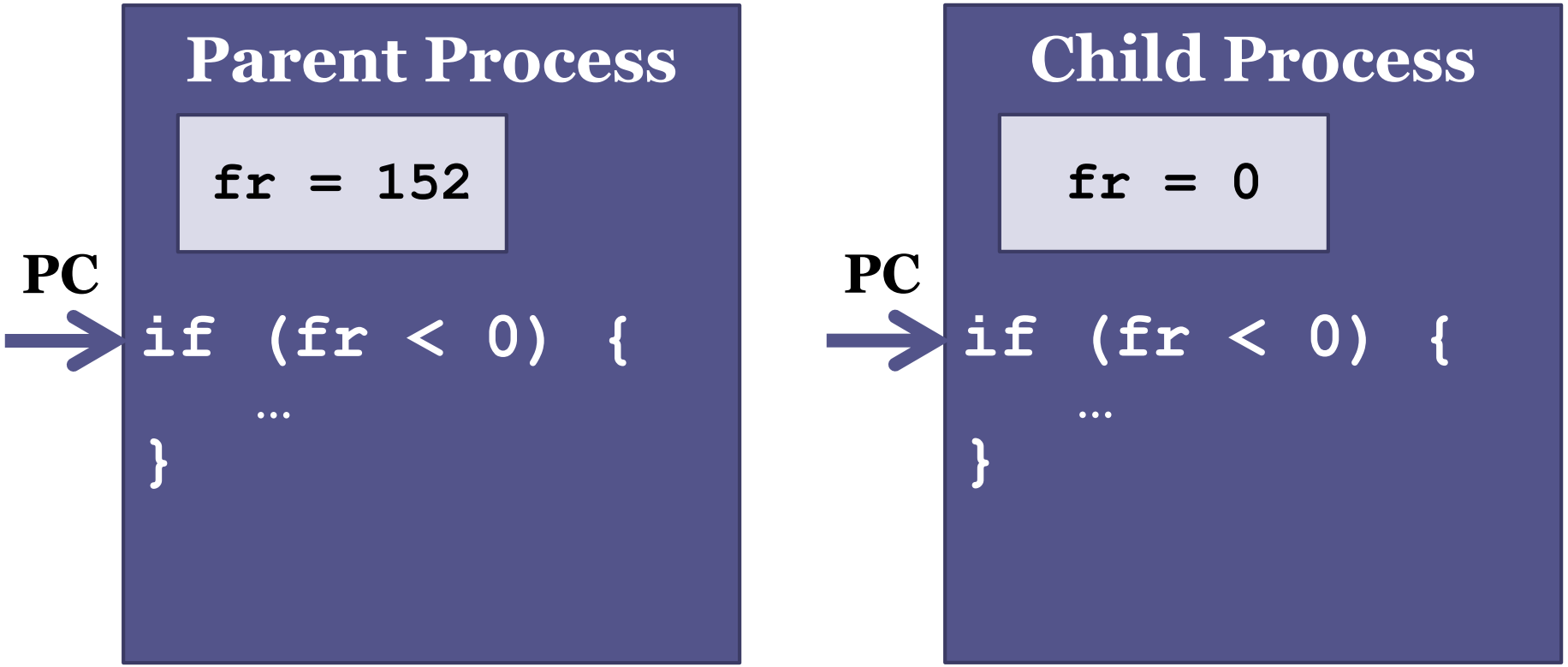
If successful fork() returns:

- 0 in fr for child process
- unique +ve id in fr for parent process

Successful Fork()

fork.c

```
fr = fork();
if (fr < 0) { …
```



fork.c

```
  else if (fr == 0) { /* child process */
     execlp("/bin/ls", "ls", NULL);
  }
  else { /* parent process */
/* parent will wait for the child to
  complete */
     wait (NULL);
     printf ("Child Complete");
     exit(0);
  }
}
```

Child Process

fork.c

```
else if (fr == 0) {
    execlp("/bin/ls", "ls", NULL);
 }
```

- Belongs to the exec() family
- Replaces calling process image by “ls”
- NULL terminates the list of arguments

Parent Process

fork.c

```
else { /* parent process */
/* parent will wait for the child
  to complete */
     wait (NULL);
     printf ("Child Complete");
     exit(0);
  }
```

- wait() for children to complete
- wait(&status), return termination info in status

ex1.c

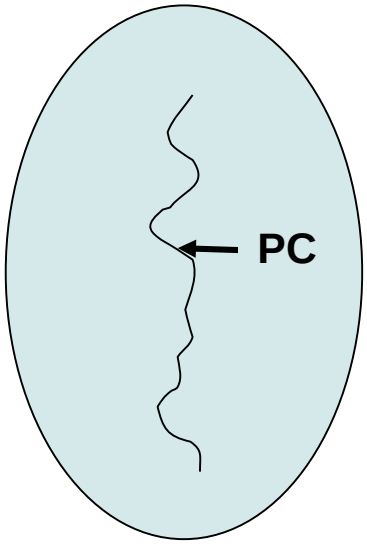
```
int main()
{
        pid_t pid;
        int toParent;
        int fromChild;

        pid = fork();

        if (pid < 0) {
                fprintf(stderr, "Fork Failed");
                exit(-1);
        }
        else if (pid == 0) {
                printf("I am the child\n");
                sleep(1); // allow parent to wait
                printf("Enter an integer between 0 and 255: ");
                scanf("%d", &toParent);
                exit(toParent);
        }
        else {
                printf("I am the parent\n");
                wait(&fromChild);
                printf("Child Completed with %d \n", WEXITSTATUS(fromChild));
        }
        printf("The parent is done\n");
        exit(0);
}
```

POSIX Threads

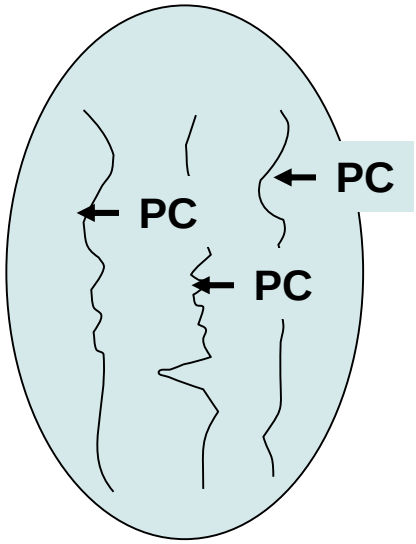
Overhead of Processes

- Processes allow the OS to overlap I/O and computation, creating an efficient system
- **Overhead**:
  - Process creation
  - Context switching
  - Swapping
  - All require kernel intervention

Threads enhance the process concept



Traditional Process



Multi-threaded Process

- A set of threads share the same address space

Advantages of Threads

- Multithreaded-programs are easy to map to multiprocessors
- It is easier to engineer applications with threads

Threads versus Processes

- Threads (same address space) are not protected from each other
  - Require care from developers
- Context switching is cheaper

Threads versus Processes

- Traditional process executes a system call, it must block
- A thread executes a system call, peer threads may still proceed

POSIX Threads

- Called also Pthreads
- A standard programming interface for threads in C
- IEEE POSIX 1003.1c standard

- A nice tutorial at : https://computing.llnl.gov/tutorials/pthreads/

pex1.c

```c
#include <pthread.h>
#include <stdio.h>


void *runner(void *param); /* thread executed function */

struct shared {
    int flag;// flags the completion of the child
    int sum;
};

struct shared s;

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

    pthread_t tid;
    pthread_attr_t attr;

    pthread_attr_init(&attr);
    pthread_create(&tid,&attr,runner,argv[1]);

    s.sum = 0;
    s.flag = 0;

    while (!s.flag)
        printf("Parent: sum is %d\n", s.sum);
    pthread_join(tid,NULL);
    printf("Parent: Finally sum is %d\n", s.sum);
}

void *runner(void *param)
{
    int i, upper = atoi(param);

    for (i = 0; i <= upper; i++)
    {
        s.sum +=i;
        printf("    Child: sum is %d\n", s.sum);
    }
    s.flag = 1;
    pthread_exit(0);
}
```

pex1.c

```
#include <pthread.h>
#include <stdio.h>

void *runner(void *param); /* thread executed
  function */

struct shared {
  int flag; // flags the completion of the child
  int sum;
};

struct shared s; // shared between parent and
  child threads
```

pex1.c

```
void *runner(void *param)
{
 int i, upper = atoi(param);
 for (i = 0; i <= upper; i++)
 {
    s.sum +=i;
    printf("    Child: sum is %d\n", s.sum);
 }
 s.flag = 1; // I am done
 pthread_exit(0);
}
```

pex1.c

```
int main (int argc, char *argv[])
{
  pthread_t tid;
  pthread_attr_t attr;

  pthread_attr_init(&attr); \\get default attributes
  pthread_create(&tid, &attr, runner, argv[1]);

  s.sum = 0;
  s.flag = 0;

  while (!s.flag)
     printf("Parent: sum is %d\n", s.sum);
  pthread_join(tid,NULL);
  printf("Parent: Finally sum is %d\n", s.sum);
}
```

Mixing C and Assembly

Mixing C and Assembly

```
__asm__ ( "assembly code"
        : output operands /* optional */
        : input operands /* optional */
        );
```

assembly.c

```c
#include <stdio.h>

int main() {

    int n1, n2, add, sub, mul, quo, rem ;

    printf( "Enter two integers:" );
    scanf( "%d%d", &n1, &n2 );

    __asm__ ( "addl %%ebx, %%eax;" : "=a" (add) : "a" (n1) , "b" (n2) );
     printf( "%d + %d = %d\n", n1, n2, add );

    __asm__ ( "subl %%ebx, %%eax;" : "=a" (sub) : "a" (n1) , "b" (n2) );
     printf( "%d - %d = %d\n", n1, n2, sub );

    __asm__ ( "imull %%ebx, %%eax;" : "=a" (mul) : "a" (n1) , "b" (n2) );
    printf( "%d * %d = %d\n", n1, n2, mul );

    return 0 ;
}
```