# Programing with Multiple Processes in C

fork, wait, execlp, file operations, and make
Help Session 2: Spring 2022

# Assignment Information

- Four executables will be needed
  - **Generator** – Main program, that opens, reads the characters and closes the file, forks child processes.
  - **Lucas, HexagonalSeries, HarmonicSeries**

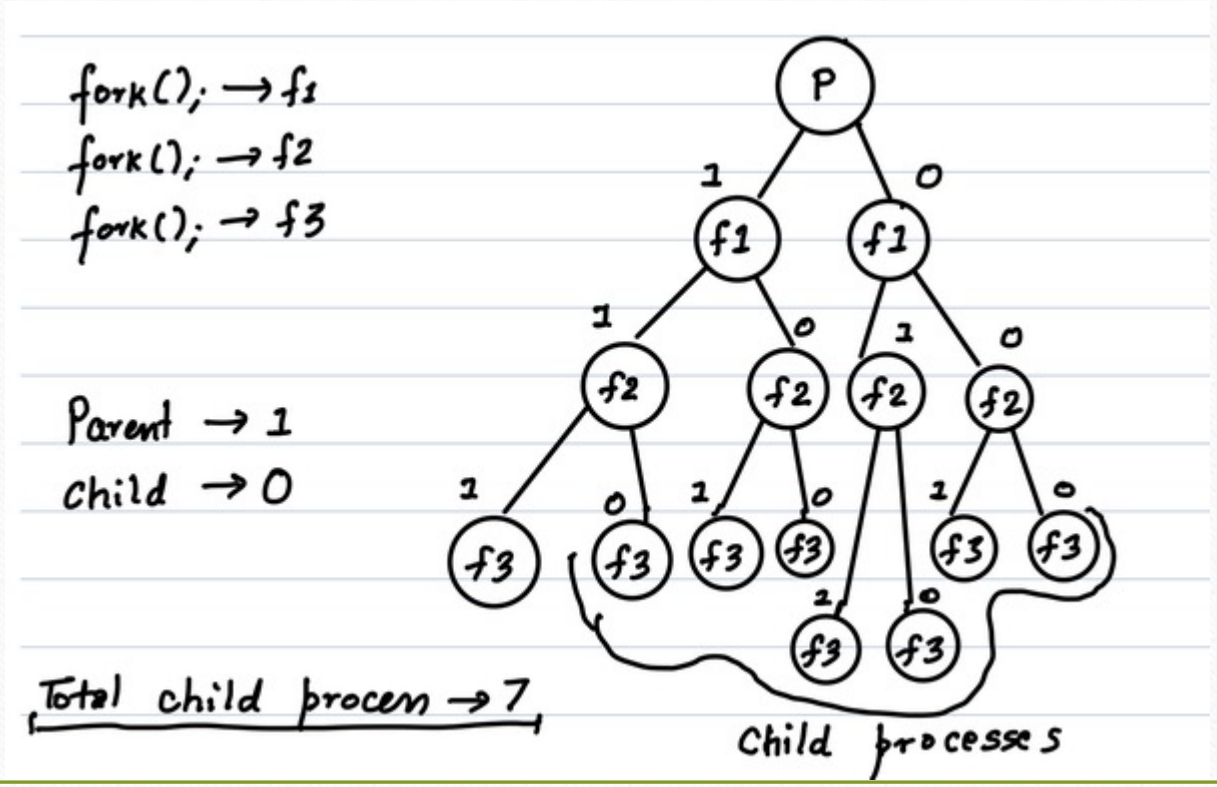
# Outline

- Learn how to use the following
  - fork()
  - wait()
  - execlp()
  - file operations
  - make

# fork()

- Generates an exact copy of parent process except for the value it returns.
- Both Processes continue to work after the fork() execution.
- In a child process, fork() returns zero
- In the parent process it will return the child's process ID
- If return value is -1, then fork() failed.
- Any process can retrieve its process ID with getpid().
- Syntax:
  - pid_t pid=fork();

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main() {
fork();
fork();
fork();
printf("hello\n");
return 0;
}
```



# wait()

- Makes parent process wait until the child has been entirely executed .
- Use WIFEXITED() to check whether child process has terminated normally, as opposed to dying with a signal .
- Use WEXITSTATUS() to retrieve return value of child process
- Syntax: pid_t wait(int *stat_loc);

# execlp()

- Executes a new program within a child process
- Arguments passed - the name of the executable and filename like "./Starter", ”Starter"
- Also pass any needed command line arguments as parameters
- Terminate list of arguments with NULL
- Syntax
  - *int execlp( const char * file, const char * arg0, const char * arg1, … const char * argn, NULL );*

# File Operations

- We need several functions for this assignment.
- They are:
  - fopen()
  - fclose()
  - fgets() or fgetc()

# fopen()

- Used to open a file, whose name is given as the argument.
- It returns a pointer to the opened file.
- Syntax:
  - FILE * fp = fopen(const char *filename, const char *mode)

# fclose()

- Closes the stream to the file.
- Buffers are flushed.
- Syntax
  - int fclose(FILE *stream)

# fgets()

- Reads a line from a file
- Puts the line into the provided array/string
- Syntax:

```
int fgets(char *s, int size, FILE *stream)
```

- Use:

```
char buf[256];
while (fgets(buf, sizeof(buf), in)
        // deal with the string in buf
```

# Why use make?

- Enables developers to easily compile large and complex programs with many components.
- Situation: There are thousands of lines of code, distributed in multiple source files, written by many developers and arranged in several sub-directories. This project also contains several component divisions and these components may have complex inter-dependencies.

# Demo Makefile
# Simple version

```
CC = gcc

default: all
all: p1 p2 p3 p4

.c.o:
	$(CC) -o P1.c P2.c P3.c p4.c

package:
	zip -r Mohit.zip p1.c p2.c p3.c p4.c Makefile input.txt

clean:
	rm -f *.o *~ p1 p2 p3 p4
```

Defining the macros

Convert all c files to o files

Compress the file

Clean the unnecessary files

# Variable assignments in make

- By convention, predefined variable names used in a Makefile are in upper case, and user-defined variables are lower case.

  Example: CC = gcc

- We can use the value assigned later as $()

  Example: $(CC)

# Makefile Structure

- Makefile contains definitions and rules.
- A definition has the form:

  VAR = value

- A rule has the form:

  Output files: input files

  <tab>Commands to turn inputs to outputs

- All commands must be tab-indented. Spaces don't work!
- The make <target> command executes the rule with the <target>. If target not is specified, it defaults to the first rule defined in the Makefile.

# Patterns and Special variables

- % : Wildcard pattern-matching, for generic targets.
- $@ : Full target name of the current target.
- $? : Returns the dependencies that are newer than the current target.
- $* : Returns the text that corresponds to % in the target.
- $< : Name of the first dependency.
- $^ : Name of the all dependencies with space as the delimiter.

# Demo Makefiles

```
CC= gcc

build : output input

output : output.c input
	$(CC) output.c -o $@

input : input.c
	$(CC) input.c -o $@


clean:
	rm -f *.o output input

test:
	./output 10
```

```
# List of files
C_SRCS          = Initiator.c Worker.c
C_OBJS          = Initiator.o Worker.o
C_HEADERS       = Worker.h

OBJS            = ${C_OBJS}
EXE             = Initiator
DEFINES         = -DDEBUG

# Compiler and loader commands and flags
GCC             = gcc
GCC_FLAGS       = -g -std=c11 -Wall -c
LD_FLAGS        = -g -std=c11 -Wall

# Compile .c files to .o files
.c.o:
	$(GCC) $(GCC_FLAGS) $(DEFINES) $<

# Target is the executable
default: $(OBJS)
	$(GCC) $(LD_FLAGS) $(OBJS) -o $(EXE)

# Recompile C objects if headers change
${C_OBJS}:      ${C_HEADERS}

# Clean up the directory
clean:
	rm -f *.o *~ $(EXE)
```

# Quiz 2 Questions

Q: A routine responding to a system call can be executed in either user mode or kernel mode.

a. True

b. False

# Quiz 2 Questions

Q: Registers and cache memories are generally not on the same chip in modern processors.

a. True

b. False

# Thank You

# Acknowledgements


- These slides are based on contributions of current and past CS370 instructors and TAs, including J. Applin,A. Yeluri, Y. K. Malaiya, Phil sharp and S. Pallickara.