lecture 12

- *handling exceptions reminder (start using try, throw, catch please)*
- *Linux (UNIX) processes*
  - *memory model*
  - *fork, exec, wait …*
- *I/O -input, output*
  - *creating,writing, reading, deleting files*

Kenneth J. Roche
University of Washington
AMATH 483 / 583 High Performance Scientific Computing
Spring Quarter 2025

handling exceptions in C++

- exception: C++ function encounters situation it cannot recover from
  - idea: inform the caller the function failed
    - *#include <exception>* and keyword ***throw***

```cpp
#include <vector>
#include <exception>
void daxpy(double a, const std::vector<double> &x, std::vector<double> &y)
{
    int n = x.size();
    if (y.size() != n)
    {
        throw std::invalid_argument("Vector dimensions do not match");
    }

    for (int i = 0; i < n; i++)
    {
        y[i] = a * x[i] + y[i];
    }
}
```

Kenneth J. Roche
University of Washington

handling exceptions in C++

- caller can catch the exception signal and respond
  - keywords ***try , catch***

```
//  test1, 2x1 + 3x1 fails
std::vector<double> w = {0., 1., 2.};
try
{
    daxpy(aa, x, w);
}
catch (const std::invalid_argument &e)
{
    std::cout << "Exception caught: test 1 " << e.what() << std::endl;
    l3_err_cnt++;
}
// do something about it
if (l3_err_cnt != 1)
{
    std::cout << "\tincorrect L1 error returns for daxpy() index testing: " << l3_err_cnt
    << " out of 3" << std::endl;
}
else
    std::cout << "\tcorrect L1 error returns for daxpy() index testing!" << std::endl;
```

Unix - like process (a program executing in memory)

Kenneth J. Roche
University of Washington
AMATH 483 / 583 High Performance Scientific Computing
Spring Quarter 2025

process, a program executing in memory

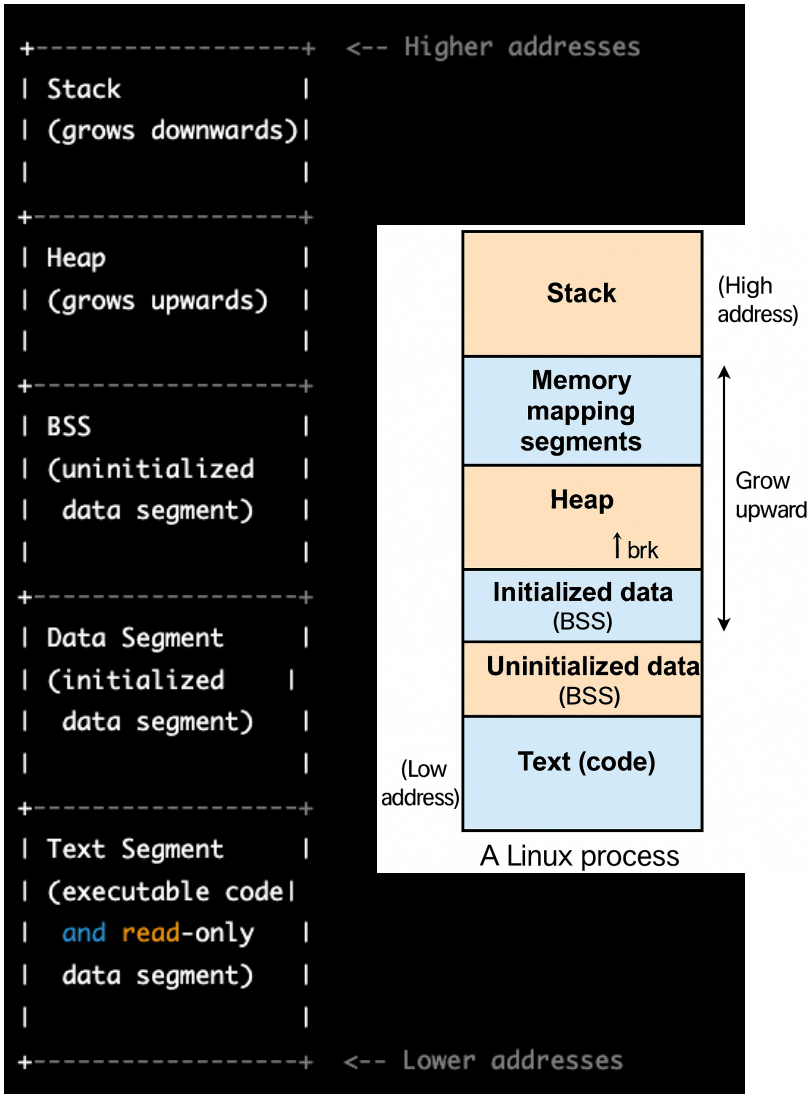
stack, located at the top of the process's memory space and grows downwards

heap, located at the bottom of the memory space and grows upwards

BSS (block started by symbol) segment, contains uninitialized global and static data

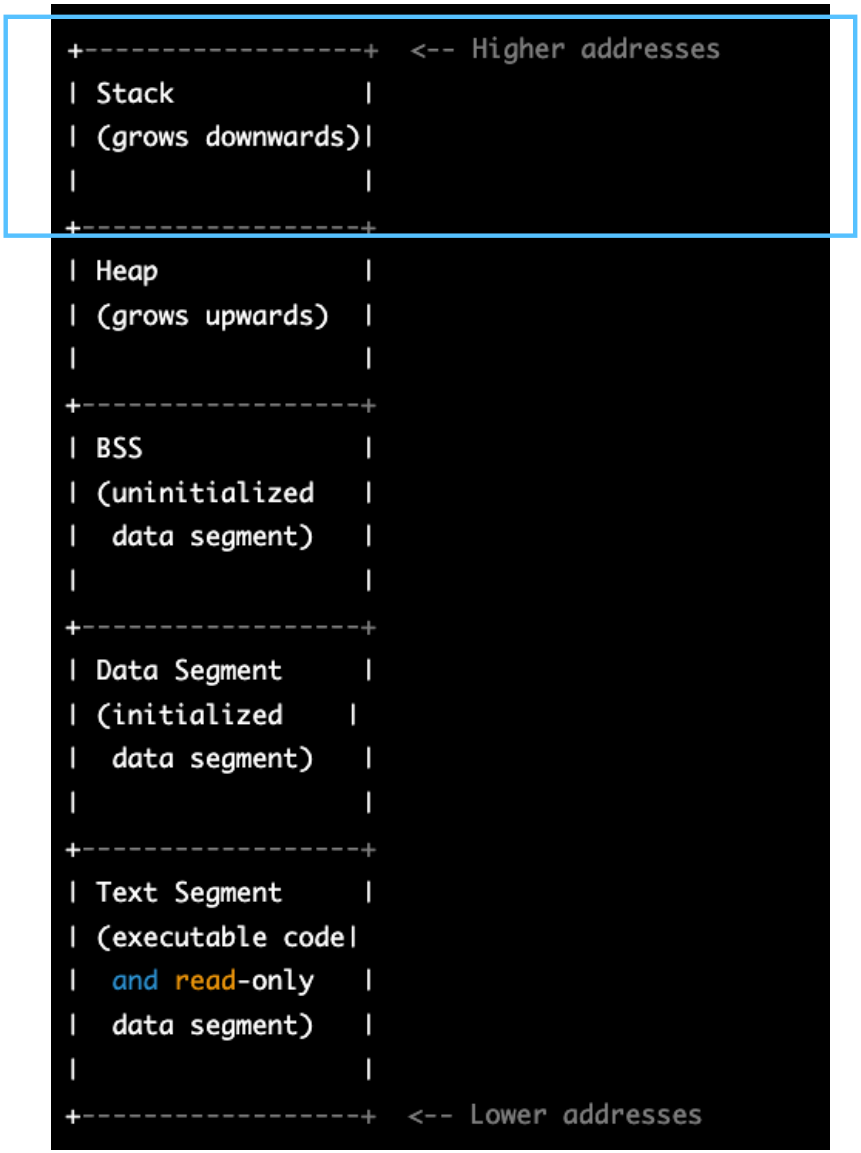
data segment contains initialized data

text segment, contains the executable code and read-only data



Kenneth J. Roche
University of Washington
AMATH 483 / 583 High Performance Scientific Computing
Spring Quarter 2025

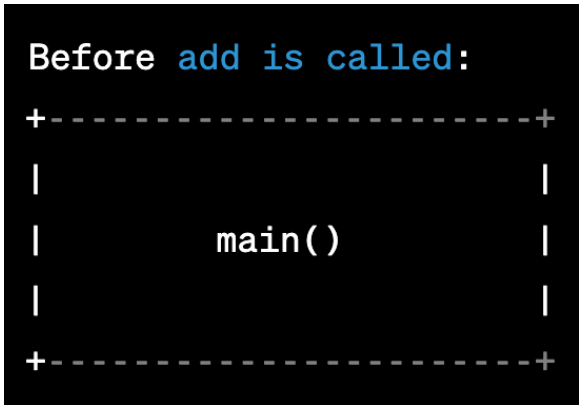
- **stack** is a segment of a process's memory space: store local variables, function call frames, and other data related to function calls
- **function call**: fnc parameters and local variables are allocated on the stack. As the function executes, its local variables and function call frames are **push**ed on the stack, and when the function returns, they are **pop**ped off the stack
- most recently pushed data is located at the top of the stack, and the oldest data is located at the bottom of the stack (**LIFO**)
- also stores return addresses, which are used to return control to the caller function after a function call is completed



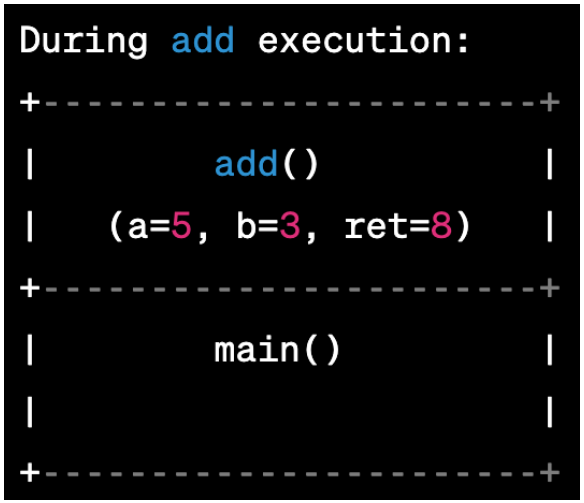
Kenneth J. Roche
University of Washington
AMATH 483 / 583 High Performance Scientific Computing
Spring Quarter 2025

```cpp
#include <iostream>
using namespace std;

int add(int a, int b) {
    return a + b;
}

int main() {
    int x = 5;
    int y = 3;
    int z = add(x, y);
    cout << "The sum of " << x << " and " << y << " is " << z << endl;
    return 0;
}
```

Before add is called:



During add execution:

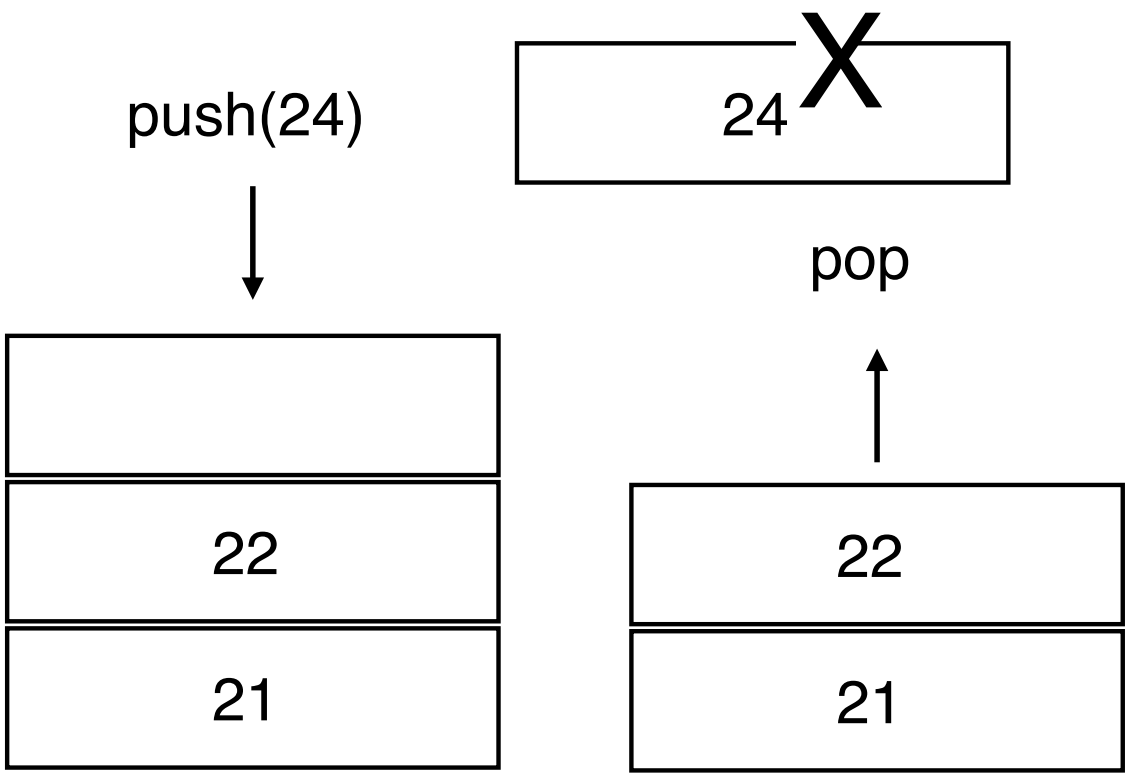


- initially the call stack contains only the activation record for **main** function
- when **add** called w/ *a=5 and b=3*, a new activation record *pushed* to top of stack
  - record contains values of a and b, and space (*ret*) for the return value
  - value 8 is stored in *ret* space of the add activation record
- when **add** returns, its activation record is removed from the top of the call stack
- execution flow control returns to **main**

Kenneth J. Roche
University of Washington

Kenneth J. Roche
University of Washington
AMATH 483 / 583 High Performance Scientific Computing
Spring Quarter 2025

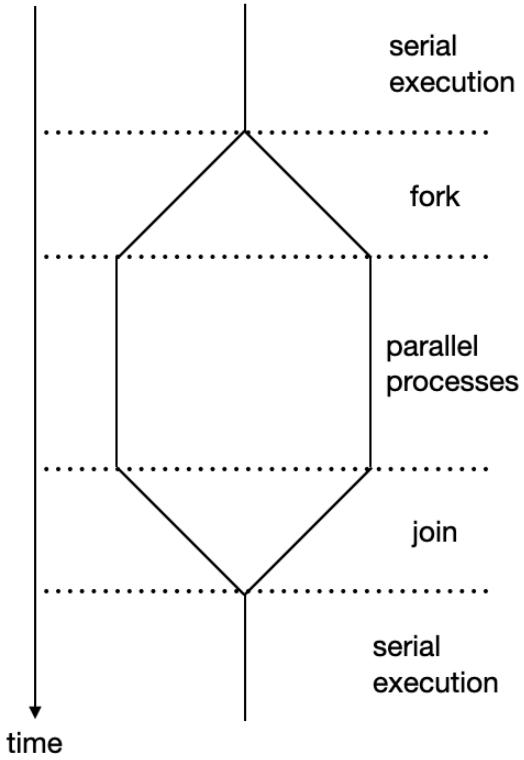
standard template library ***stack***




```cpp
#include <iostream>
#include <stack>
using namespace std;
int main() {
    stack<int> stack;
    stack.push(21);
    stack.push(22);
    stack.push(24);
    stack.push(25);

    stack.pop();
    stack.pop();

    while (!stack.empty()) {
      cout << stack.top() <<" ";
      stack.pop();
    }

    cout << endl;
}
```

```
[WE42365:~/Desktop/kr-code-training/cplusplus] d3y402% g++ -std=c++14 -o xstack stackiter.cpp
[WE42365:~/Desktop/kr-code-training/cplusplus] d3y402% ./xstack
22 21
```

process creation (from within a process)




```
pid_t fork(void);

int exec(const char *path, char *const argv[]);

int execvp(const char *file, char *const argv[]);

pid_t wait(int *status);
```

fork: *creates a new process during process execution*

- *both child and parent processes continue executing from the point in the code of the fork*
- *now parent and child processes have different PIDs and memory spaces*

```c
#include <unistd.h>

pid_t fork(void);
```

- **Creates a new process** (child) by duplicating the calling process.
- Returns:
  - `> 0`: PID of the child in the parent.
  - `0`: In the child.
  - `< 0`: Error occurred (no child created).

***exec(vp)**: replaces current process image with a new process image during process execution which will be specified by a file path and list of arguments corresponding to an executable file (execvp searches PATH)*

- *new process inherits calling processes environment, file descriptors, and signal handlers*
- *only return if they fail / have error*

`execl()`

```c
#include <unistd.h>

int execl(const char *path, const char *arg0, ..., (char *)NULL);
```

- Takes a variable number of arguments ending with `NULL`.
- Example: `execl("/bin/ls", "ls", "-l", NULL);`

`execvp()` — Searches `PATH`

```c
#include <unistd.h>

int execvp(const char *file, char *const argv[]);
```

- Like `execv()` but searches in `$PATH`.
- Most commonly used.
- Example: `char *args[] = {"ls", "-l", NULL}; execvp("ls", args);`

`execv()`

```c
#include <unistd.h>

int execv(const char *path, char *const argv[]);
```

- `argv` is a NULL-terminated array of strings.
- Example: `char *args[] = {"ls", "-l", NULL}; execv("/bin/ls", args);`

***wait**: suspends the calling process (parent) until any one of it's child processes terminates*

`wait()`

```c
#include <sys/types.h>
#include <sys/wait.h>

pid_t wait(int *status);
```

- Waits for **any child** to terminate.
- Returns PID of the child; `status` can be decoded with macros:
  - `WIFEXITED(status)`
  - `WEXITSTATUS(status)`
  - `WIFSIGNALED(status)`
  - `WTERMSIG(status)`

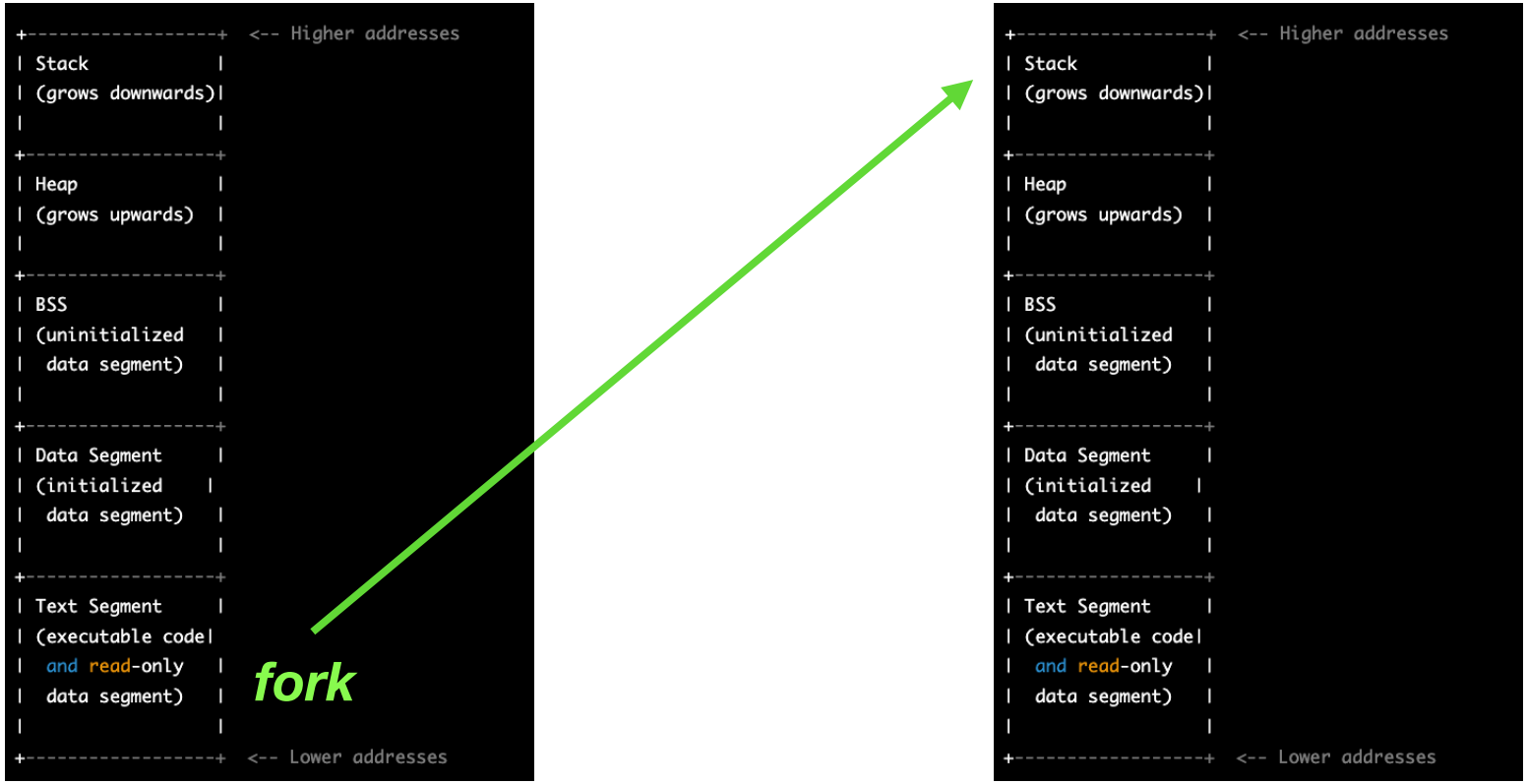
`waitpid()`

```c
#include <sys/types.h>
#include <sys/wait.h>

pid_t waitpid(pid_t pid, int *status, int options);
```

- Waits for a **specific child**.
- `pid` : child's PID or `-1` to wait for any child.
- `options` : use `0` or macros like `WNOHANG` .

parent

child



Kenneth J. Roche
University of Washington
AMATH 483 / 583 High Performance Scientific Computing
Spring Quarter 2025

Child Process:

- After `fork()` is called, the child process executes the same code as the parent, starting from the point of the `fork()` call.
- The child process can modify its own memory space independently of the parent process. However, any changes made by the child process do not affect the parent process.
- When `execvp()` is called in the child process, the child process's memory space is replaced with the new program's code, data, and stack. This new program starts executing from the beginning of its `main()` function. The `execvp()` system call does not return, unless there is an error.

Parent Process:

- After the child process is created with `fork()`, the parent process continues executing the same code as before the `fork()` call.
- The parent process can use the `wait()` system call to wait for the child process to complete. During this time, the child process continues executing independently of the parent process.
- Once the child process completes, the parent process continues executing from the point where the `wait()` call was made.

```cpp
#include <iostream>
#include <unistd.h>   // fork, execvp
#include <sys/wait.h> // waitpid
#include <cstdlib>    // exit

int main()
{
    pid_t pid = fork(); // Create child process

    if (pid < 0)
    {
        std::cerr << "Fork failed!" << std::endl;
        return 1;
    }

    if (pid == 0)
    {
        // --------- CHILD PROCESS ---------
        std::cout << "[Child] PID: " << getpid() << ", PPID: " << getppid() << std::endl;

        // Command: ls -l
        char *args[] = {(char *)"ls", (char *)"-l", nullptr};

        std::cout << "[Child] Executing `ls -l` using execvp()..." << std::endl;
        execvp("ls", args);

        // If execvp returns, an error occurred
        std::cerr << "[Child] execvp() failed!" << std::endl;
        exit(1);
    }
    else
    {
        // --------- PARENT PROCESS ---------
        std::cout << "[Parent] PID: " << getpid() << ", created child with PID: " << pid << std::endl;

        int status;
        pid_t wpid = waitpid(pid, &status, 0); // Wait for child

        if (wpid == -1)
        {
            std::cerr << "[Parent] Error while waiting for child!" << std::endl;
            return 1;
        }

        if (WIFEXITED(status))
        {
            std::cout << "[Parent] Child exited with status " << WEXITSTATUS(status) << std::endl;
        }
        else if (WIFSIGNALED(status))
        {
            std::cout << "[Parent] Child was killed by signal " << WTERMSIG(status) << std::endl;
        }
        else
        {
            std::cout << "[Parent] Child terminated abnormally." << std::endl;
        }
    }

    return 0;
}
```

Kenneth J. Roche
University of Washington
AMATH 483 / 583 High Performance Scientific Computing
Spring Quarter 2025

```cpp
#include <iostream>
#include <unistd.h>   // fork, execvp
#include <sys/wait.h> // waitpid
#include <cstdlib>    // exit

int main()
{
    pid_t pid = fork(); // Create child process

    if (pid < 0)
    {
        std::cerr << "Fork failed!" << std::endl;
        return 1;
    }

    if (pid == 0)
    {
        // --------- CHILD PROCESS ---------
        std::cout << "[Child] PID: " << getpid() << ", PPID: " << getppid() << std::endl;

        // Command: ls -l
        char *args[] = {(char *)"ls", (char *)"-l", nullptr};

        std::cout << "[Child] Executing `ls -l` using execvp()..." << std::endl;
        execvp("ls", args);

        // If execvp returns, an error occurred
        std::cerr << "[Child] execvp() failed!" << std::endl;
        exit(1);
    }
    else
    {
        // --------- PARENT PROCESS ---------
        std::cout << "[Parent] PID: " << getpid() << ", created child with PID: " << pid << std::endl;

        int status;
        pid_t wpid = waitpid(pid, &status, 0); // Wait for child

        if (wpid == -1)
        {
            std::cerr << "[Parent] Error while waiting for child!" << std::endl;
            return 1;
        }

        if (WIFEXITED(status))
        {
            std::cout << "[Parent] Child exited with status " << WEXITSTATUS(status) << std::endl;
        }
        else if (WIFSIGNALED(status))
        {
            std::cout << "[Parent] Child was killed by signal " << WTERMSIG(status) << std::endl;
        }
        else
        {
            std::cout << "[Parent] Child terminated abnormally." << std::endl;
        }
    }

    return 0;
}
```

```
bash-3.2$ g++ -std=c++17 -o xprocess1 process1.cpp
bash-3.2$ ./xprocess1
[Parent] PID: 86454, created child with PID: 86456
[Child] PID: 86456, PPID: 86454
[Child] Executing `ls -l` using execvp()...
total 11312
-rw-r--r--@ 1 kennethroche  staff      563 Apr 20 23:09 automobile.hpp
-rw-r--r--@ 1 kennethroche  staff     2246 Apr  6 22:53 basic-operations.cpp
-rw-r--r--@ 1 kennethroche  staff      435 Apr 20 23:08 car.hpp
-rw-r--r--@ 1 kennethroche  staff     1417 Apr 13 20:26 class-access.cpp
-rw-r--r--  1 kennethroche  staff    12632 Apr 13 20:49 class-access.o
-rw-r--r--@ 1 kennethroche  staff     1080 Apr  7 08:46 cpp-types.cpp
-rw-r--r--@ 1 kennethroche  staff      895 Apr 15 22:52 dyn_mem1.cpp
-rw-r--r--@ 1 kennethroche  staff      548 Apr 15 23:07 dyn_mem_fnc.cpp
-rwxr-xr-x  1 kennethroche  staff    40256 Apr 16 00:00 xmemcpy-dp-shl
-rwxr-xr-x  1 kennethroche  staff    42536 Apr 16 15:07 xmeps
-rwxr-xr-x  1 kennethroche  staff    40472 Apr 15 21:39 xmystorage
-rwxr-xr-x  1 kennethroche  staff    40344 Apr 25 00:20 xprocess1
-rwxr-xr-x  1 kennethroche  staff   128408 Apr 17 22:23 xvadd1
-rwxr-xr-x  1 kennethroche  staff   128408 Apr 17 22:34 xvadd2
[Parent] Child exited with status 0
bash-3.2$
```

Kenneth J. Roche
University of Washington
AMATH 483 / 583 High Performance Scientific Computing
Spring Quarter 2025

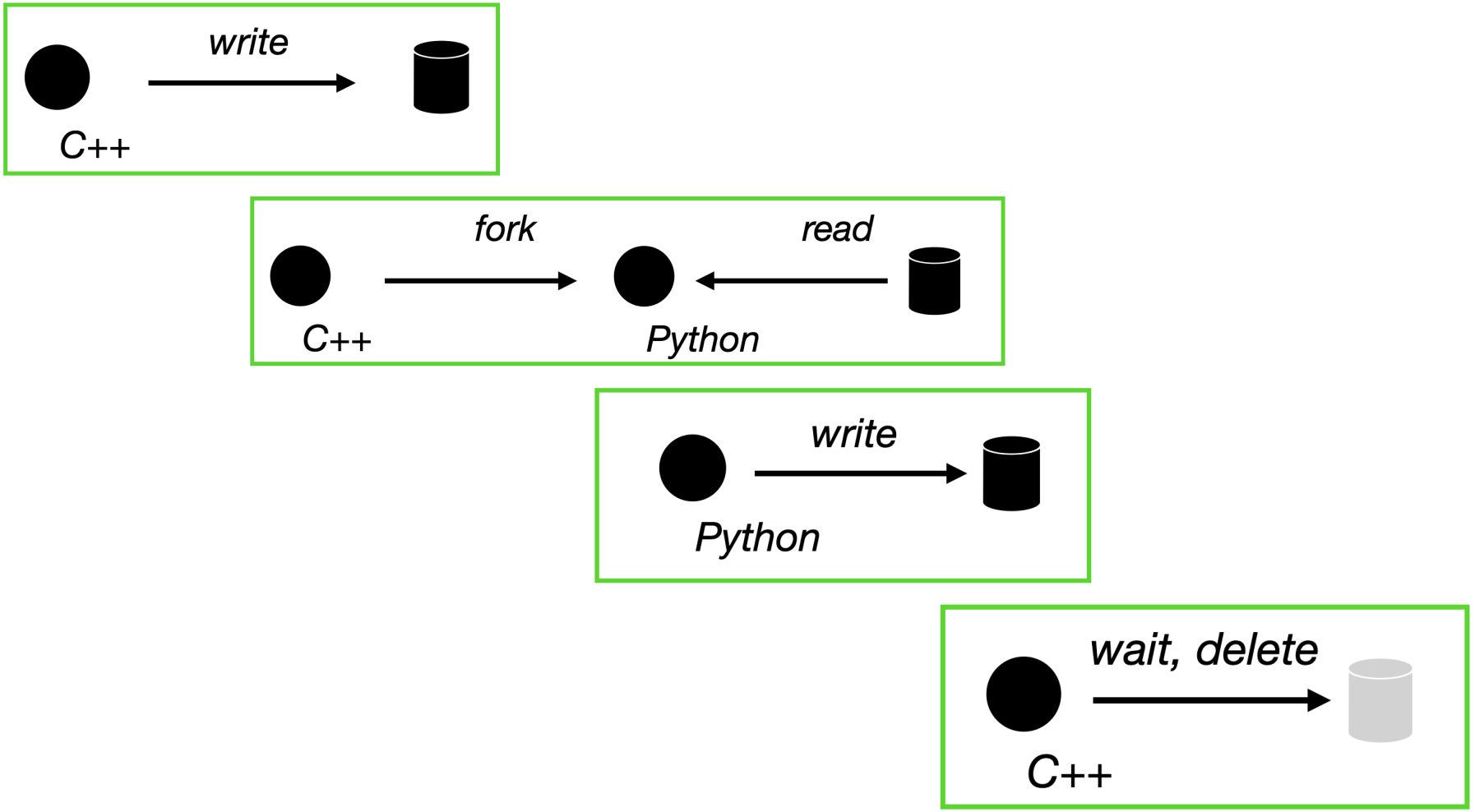
```cpp
#include <iostream>
#include <unistd.h>
#include <string>
#include <sys/wait.h>

using namespace std;

int main() {
    pid_t pid;
    int status;

    pid = fork();

    if (pid == 0) {
        cout << "This is the child process." << endl;

        const char* args[] = { "ls", nullptr };
        execvp(args[0], (char**)args);

        cout << "This should not be printed." << endl;
    }
    else if (pid > 0) {
        cout << "This is the parent process." << endl;
        wait(&status);
        cout << "Child process exited with status " << status << "." << endl;
    }
    else {
        cout << "Error: fork() failed." << endl;
        return 1;
    }

    return 0;
}
```

Kenneth J. Roche
University of Washington
AMATH 483 / 583 High Performance Scientific Computing
Spring Quarter 2025

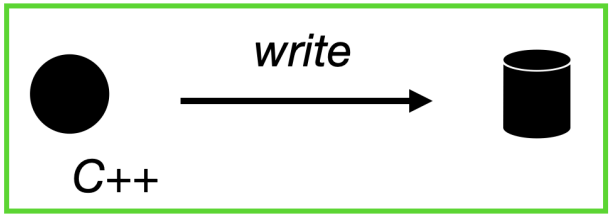
```
[WE42365:~/Desktop/kr-code-training/cplusplus] d3y402% g++ -o xforkexample process-creation.cpp
[WE42365:~/Desktop/kr-code-training/cplusplus] d3y402% ./xforkexample
This is the parent process.
This is the child process.
total 17520
-rw-r--r--@  1 d3y402  PNL\Domain Users     2910 Apr 22 22:49 README.txt
-rw-r--r--@  1 d3y402  PNL\Domain Users      116 Apr 16 21:38 addition.cpp
-rw-r--r--@  1 d3y402  PNL\Domain Users      111 Apr 16 21:53 addition.hpp
-rw-r--r--@  1 d3y402  PNL\Domain Users   205008 Apr 16 21:54 addition.o
```

```
-rwxr-xr-x@  1 d3y402  PNL\Domain Users    57032 Apr 19 16:00 xtestrefblas
-rwxr-xr-x@  1 d3y402  PNL\Domain Users    97856 Apr 20 01:32 xtestrefdblas
-rwxr-xr-x@  1 d3y402  PNL\Domain Users    94624 Apr 20 01:32 xtestreftblas
-rwxr-xr-x@  1 d3y402  PNL\Domain Users   132984 Apr 17 00:06 xtesttemplates
-rwxr-xr-x@  1 d3y402  PNL\Domain Users    70536 Apr 15 18:16 xtimer
Child process exited with status 0.
[WE42365:~/Desktop/kr-code-training/cplusplus] d3y402%
```

example: (scratch the itch … plotting!)



example: (scratch the itch … plotting!?)

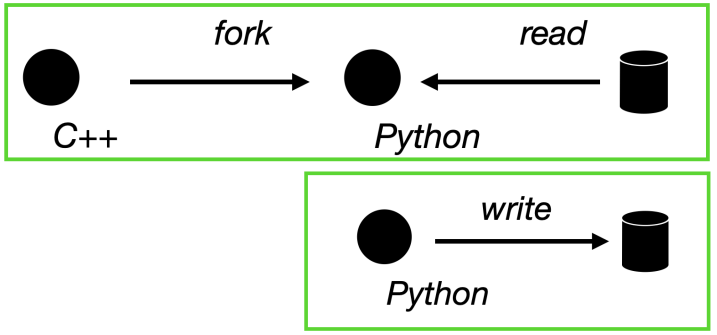



```cpp
#include <iostream>
#include <fstream>
#include <cmath>
#include <unistd.h>   // fork, execvp
#include <sys/wait.h> // waitpid
#include <cstdio>     // remove

const int NUM_POINTS = 100;
const double PI = 3.14159265358979323846;

int main()
{
    // 1. Create and write to data.txt
    std::ofstream outfile("data.txt");
    if (!outfile)
    {
        std::cerr << "Failed to open data.txt for writing!" << std::endl;
        return 1;
    }

    double xmin = -2 * PI;
    double xmax = 2 * PI;

    for (int i = 0; i < NUM_POINTS; ++i)
    {
        double x = xmin + i * (xmax - xmin) / (NUM_POINTS - 1);
        outfile << x << ", " << std::cos(x) << "\n";
    }
    outfile.close();
    std::cout << "[C++] Wrote cosine data to data.txt\n";
```

Kenneth J. Roche
University of Washington

example: (scratch the itch … plotting!?)

```
32      // 2. Fork and run Python script
33      pid_t pid = fork();
34
35      if (pid < 0)
36      {
37          std::cerr << "Fork failed!" << std::endl;
38          return 1;
39      }
40
41      if (pid == 0)
42      {
43          // --- CHILD ---
44          std::cout << "[Python] Launching plot_data.py...\n";
45          char *args[] = {(char *)"python3", (char *)"plot_data.py", nullptr};
46          execvp("python3", args);
47
48          std::cerr << "[Child] execvp() failed!\n";
49          return 1;
50      }
51      else
52      {
53          // --- PARENT ---
```

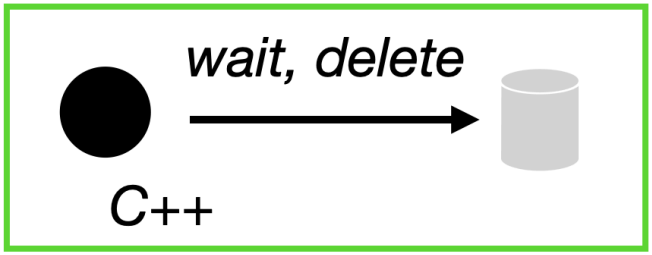



```
import numpy as np
import matplotlib.pyplot as plt

# 1. Load the data
data = np.loadtxt("data.txt", delimiter=',')
x = data[:, 0]
y = data[:, 1]

# 2. Plot
plt.figure(figsize=(8, 4))
plt.plot(x, y, label=r'$\cos(\theta)$', color='blue')
plt.xlabel(r'$\theta$ (radians)')
plt.ylabel(r'$\cos(\theta)$')
plt.title('Cosine Function')
plt.grid(True)
plt.legend()
plt.tight_layout()

# 3. Save
plt.savefig("cosine_plot.png")
print("[Python] Saved cosine_plot.png")
```

Kenneth J. Roche
University of Washington

example: (scratch the itch … plotting!?)

```
51      else
52      {
53          // --- PARENT ---
54          int status;
55          waitpid(pid, &status, 0);
56          std::cout << "[C++] Python script finished.\n";
57
58          // 3. Delete data.txt
59          if (std::remove("data.txt") == 0)
60          {
61              std::cout << "[C++] Deleted data.txt\n";
62          }
63          else
64          {
65              std::cerr << "[C++] Failed to delete data.txt\n";
66          }
67      }
68
69      return 0;
70  }
```



Kenneth J. Roche
University of Washington

End Lecture 12

Kenneth J. Roche
University of Washington
AMATH 483 / 583 High Performance Scientific Computing
Spring Quarter 2025