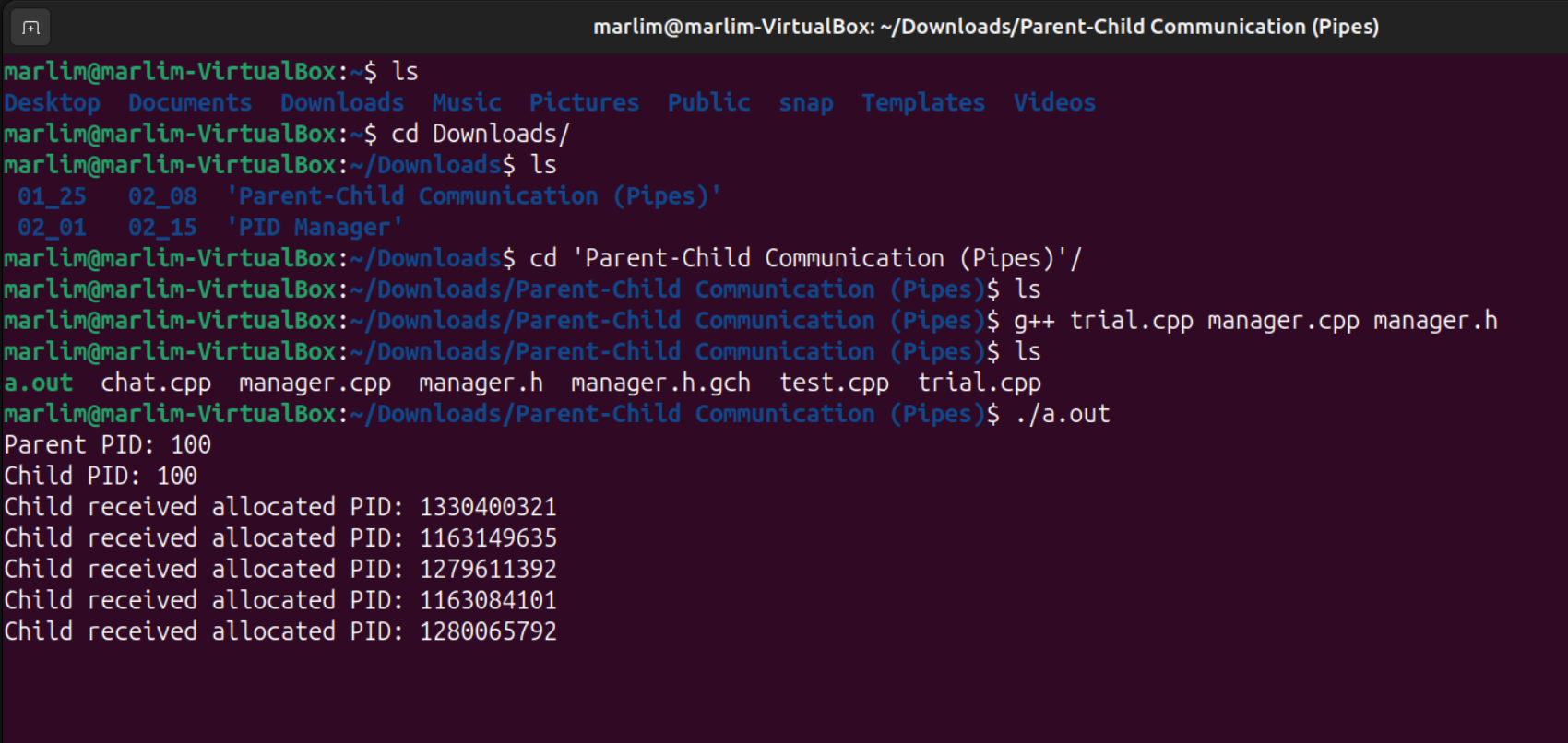
Output



Note: Unfortunately I was unable to get the pipes to properly work. The PID did not pass properly to the child and I could not get it back to try and release the PID in the parent (so it was not able to exit cleanly).

Main.cpp

#include <iostream>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <cstring>

#include <string>

#include "manager.h"

#define BUFFER\_SIZE 25

#define READ\_END 0

#define WRITE\_END 1

int main() {

char write\_msg[BUFFER\_SIZE] = "Greetings";

char read\_msg[BUFFER\_SIZE];

// create the pipe

int fd[2];

if (pipe(fd) == -1) {

std::cerr << "Pipe failed" << std::endl;

return 1;

}

// fork a child process

pid\_t pid = fork();

if (pid < 0) {

std::cerr << "Fork failed!" << std::endl;

return 1;

} else if (pid > 0) {

// Parent process

PidManager ParentManager;

if (ParentManager.allocate\_map() == -1) {

std::cerr << "Failed to allocate bitmap." << std::endl;

return 1;

}

int parent\_pid = ParentManager.allocate\_pid();

if (parent\_pid == -1) {

std::cerr << "Failed to allocate PID for parent process." << std::endl;

return 1;

}

std::cout << "Parent PID: " << parent\_pid << std::endl;

while (true) {

// Wait for Child Process to issue allocation requests or send end signal

wait(NULL);

read(fd[READ\_END], read\_msg, BUFFER\_SIZE);

if (strcmp(read\_msg, "DONE") == 0) break;

// Allocate Child PID

if (strcmp(read\_msg, "ALLOCATE") == 0) {

int child\_pid = ParentManager.allocate\_pid();

write(fd[WRITE\_END], &child\_pid, sizeof(child\_pid));

std::cout << "Parent allocated PID: " << child\_pid << std::endl;

}

// wait(NULL);

// Release Child PID

read(fd[READ\_END], read\_msg, BUFFER\_SIZE);

if (strcmp(read\_msg, "RELEASE") == 0) {

int child\_pid;

read(fd[READ\_END], &child\_pid, sizeof(child\_pid));

ParentManager.release\_pid(child\_pid);

std::cout << "Parent released PID: " << child\_pid << std::endl;

}

}

close(fd[READ\_END]);

close(fd[WRITE\_END]);

} else {

// Child process

PidManager ChildManager;

if (ChildManager.allocate\_map() == -1) {

std::cerr << "Failed to allocate bitmap." << std::endl;

return 1;

}

int child\_pid = ChildManager.allocate\_pid();

if (child\_pid == -1) {

std::cerr << "Failed to allocate PID for child process." << std::endl;

return 1;

}

std::cout << "Child PID: " << child\_pid << std::endl;

int iterations = 5;

for (int i = 0; i < iterations; i++) {

// Send "ALLOCATE" request

char allocate[] = "ALLOCATE";

write(fd[WRITE\_END], allocate, strlen(allocate) + 1);

// Wait for the parent to allocate and return the PID

// wait(NULL);

int received\_pid;

read(fd[READ\_END], &received\_pid, sizeof(received\_pid));

std::cout << "Child received allocated PID: " << received\_pid << std::endl;

// Send "RELEASE" request

char release[] = "RELEASE";

write(fd[WRITE\_END], release, strlen(release) + 1);

// wait(NULL);

write(fd[WRITE\_END], &received\_pid, sizeof(received\_pid));

// Wait for the parent to release the PID

wait(NULL);

if (i == (iterations - 1)) {

char done[] = "DONE";

write(fd[WRITE\_END], done, strlen(done) + 1);

}

}

close(fd[READ\_END]);

close(fd[WRITE\_END]);

}

return 0;

}

Manager.h (same as last time)

// manager.h

#ifndef MANAGER\_H

#define MANAGER\_H

#include <bitset>

#include <iostream>

#define MIN\_PID 100

#define MAX\_PID 1000

class PidManager {

private:

std::bitset<MAX\_PID - MIN\_PID + 1> pid\_bitmap;

bool map\_initialization = false;

public:

// Constructor

PidManager();

// Creates and initializes a data structure for representing pids; returns −1 if unsuccessful, 1 if successful

int allocate\_map();

// Allocates a PID (returns -1 if unable to allocate a PID)

int allocate\_pid();

// Releases a PID

void release\_pid(int pid);

};

#endif // MANAGER\_H

Manager.cpp (same as last time)

// PidManager.cpp

#include "manager.h"

// Constructor

PidManager::PidManager() {}

// Initializes the PID bitmap

int PidManager::allocate\_map() {

pid\_bitmap.reset(); // All PIDs are available initially

if (pid\_bitmap.any()) { // If any bit is set, the reset failed

return -1;

}

map\_initialization = true;

return 1; // Success

}

// Allocates a PID; returns -1 if unable to allocate a PID

int PidManager::allocate\_pid() {

if (!map\_initialization) {

return -1; // Attempt to allocate PID before initializing map

}

// Find an available PID by checking the bitset

for (int i = 0; i < (MAX\_PID - MIN\_PID + 1); i++) {

if (!pid\_bitmap[i]) {

pid\_bitmap.set(i); // Mark this PID as in use

int pid = MIN\_PID + i; // Map to the actual PID value

// std::cout << "Allocated PID index: " << i << " (PID: " << pid << ")" << std::endl;

return pid;

}

}

return -1; // All PIDs are in use

}

// Releases a PID

void PidManager::release\_pid(int pid) {

if (!map\_initialization) {

return; // Attempt to release before initializing map

}

if (pid < MIN\_PID || pid > MAX\_PID) {

return; // Outside scope

}

int index = pid - MIN\_PID;

if (pid\_bitmap[index]) {

pid\_bitmap.reset(index);

// std::cout << "Released PID: " << pid << std::endl;

} else {

std::cerr << "PID " << pid << " was not allocated" << std::endl;

}

}

Test.cpp (same as last time)

#include <iostream>

#include <cstdlib>

#include <ctime>

#include "manager.h"

// Constants for the PID range

#define MIN\_PID 100

#define MAX\_PID 1000

// Test error handling by calling allocate\_pid before calling allocate\_map

void test\_allocate\_pid\_before\_allocate\_map() {

PidManager pid\_manager;

int pid = pid\_manager.allocate\_pid();

if (pid == -1) {

std::cout << "Test Passed: allocate\_pid returned -1 before calling allocate\_map." << std::endl;

} else {

std::cerr << "Test Failed: allocate\_pid failed to return -1 before calling allocate\_map." << std::endl;

}

}

// Verify that allocate\_pid returns -1 if allocate\_map was not called

void test\_allocate\_pid\_no\_initialize() {

PidManager pid\_manager;

int pid = pid\_manager.allocate\_pid();

if (pid == -1) {

std::cout << "Test Passed: allocate\_pid returned -1 when bitmap is not initialized." << std::endl;

} else {

std::cerr << "Test Failed: allocate\_pid should return -1 when not bitmap is not initialized." << std::endl;

}

}

// Test releasing a PID without initializing the bitmap

void test\_release\_pid\_without\_initialize() {

PidManager pid\_manager;

pid\_manager.release\_pid(100);

std::cout << "Test Passed: Releasing PID without initializing bitmap did not cause error." << std::endl;

}

// Allocate and release PIDs in a loop

void test\_allocate\_release\_loop() {

PidManager pid\_manager;

if (pid\_manager.allocate\_map() == -1) {

std::cerr << "Error initializing the PID map." << std::endl;

return;

}

const int iterations = 1000;

for (int i = 0; i < iterations; ++i) {

int pid = pid\_manager.allocate\_pid();

if (pid == -1) {

std::cerr << "Test Failed: Allocation failed during iteration " << i << std::endl;

return;

}

pid\_manager.release\_pid(pid); // Release the PID

}

std::cout << "Test Passed: Successfully allocated and released " << iterations << " PIDs." << std::endl;

}

// Randomly allocate and release PIDs multiple times

void test\_random\_allocate\_release() {

PidManager pid\_manager;

if (pid\_manager.allocate\_map() == -1) {

std::cerr << "Error initializing the bitmap." << std::endl;

return;

}

const int iterations = 500;

srand(time(0)); // Initialize random seed

for (int i = 0; i < iterations; ++i) {

int action = rand() % 2;

if (action == 0) {

int pid = pid\_manager.allocate\_pid();

if (pid != -1) {

// std::cout << "Allocated PID: " << pid << std::endl;

pid\_manager.release\_pid(pid);

}

} else {

int random\_pid = rand() % (MAX\_PID - MIN\_PID + 1) + MIN\_PID;

pid\_manager.release\_pid(random\_pid);

// std::cout << "Released PID: " << random\_pid << std::endl;

}

}

std::cout << "Test Passed: Successfully allocated and released PIDs randomly." << std::endl;

}

// Verify that the allocated PIDs are within the specified range

void test\_pid\_range() {

PidManager pid\_manager;

if (pid\_manager.allocate\_map() == -1) {

std::cerr << "Error initializing the bitmap." << std::endl;

return;

}

const int num\_allocations = 100;

for (int i = 0; i < num\_allocations; ++i) {

int pid = pid\_manager.allocate\_pid();

if (pid == -1) {

std::cerr << "Test Failed: PID allocation failed during iteration " << i << std::endl;

return;

}

if (pid < MIN\_PID || pid > MAX\_PID) {

std::cerr << "Test Failed: Allocated PID " << pid << " is out of range." << std::endl;

return;

}

}

std::cout << "Test Passed: All allocated PIDs are within the specified range." << std::endl;

}

// Ensure released PIDs become available for allocation

void test\_pid\_reusability() {

PidManager pid\_manager;

if (pid\_manager.allocate\_map() == -1) {

std::cerr << "Error initializing the bitmap." << std::endl;

return;

}

int pid = pid\_manager.allocate\_pid();

if (pid == -1) {

std::cerr << "Test Failed: PID allocation failed." << std::endl;

return;

}

pid\_manager.release\_pid(pid);

int new\_pid = pid\_manager.allocate\_pid();

if (new\_pid == -1 || new\_pid != pid) {

std::cerr << "Test Failed: Released PID was not reused." << std::endl;

return;

}

std::cout << "Test Passed: Released PID was successfully reused." << std::endl;

}

// Attempt to allocate a PID when the range is exhausted

void test\_pid\_exhaustion() {

PidManager pid\_manager;

if (pid\_manager.allocate\_map() == -1) {

std::cerr << "Error initializing the bitmap." << std::endl;

return;

}

for (int i = MIN\_PID; i <= MAX\_PID; ++i) {

int pid = pid\_manager.allocate\_pid();

if (pid == -1) {

std::cerr << "Test Failed: PID allocation failed when there should be available PIDs." << std::endl;

return;

}

}

int pid = pid\_manager.allocate\_pid();

if (pid == -1) {

std::cout << "Test Passed: PID allocation failed after all PIDs were used." << std::endl;

} else {

std::cerr << "Test Failed: allocate\_pid should return -1 when all PIDs are in use." << std::endl;

}

}

int main() {

// Run all the tests

test\_allocate\_pid\_before\_allocate\_map();

test\_allocate\_pid\_no\_initialize();

test\_release\_pid\_without\_initialize();

test\_allocate\_release\_loop();

test\_random\_allocate\_release();

test\_pid\_range();

test\_pid\_reusability();

test\_pid\_exhaustion();

return 0;

}