Name: Rakesh Kannan
Student ID: 200535445
ECE 565 Operating Systems Design

Project 1 Report:

Q1. What is the maximum number of processes accepted by Xinu? Where is it defined?
Ans:

By default, Xinu allows a maximum of 100 processes, as specified by the constant NPROC in the **config/conf.h** file. The value for NPROC is also set in **process.h** as 8 but the **conf.h** file value overrides the value defined in **include/process.h**.

```
/* Maximum number of processes in the system */
#ifndef NPROC
#define NPROC       8
#endif

/* Configuration and Size Constants */
#define NPROC       100    /* number of user processes     */
```

Q2. What does Xinu define as an “illegal PID”?
Ans:

The isbadpid() inline function, defined in **include/process.h**, specifies the conditions for an invalid PID. A PID is considered invalid if it is

- less than zero
- greater than or equal to the maximum allowed processes (i.e., NPROC)
- if it corresponds to a process table entry that is not currently in use (i.e., PR_FREE state).

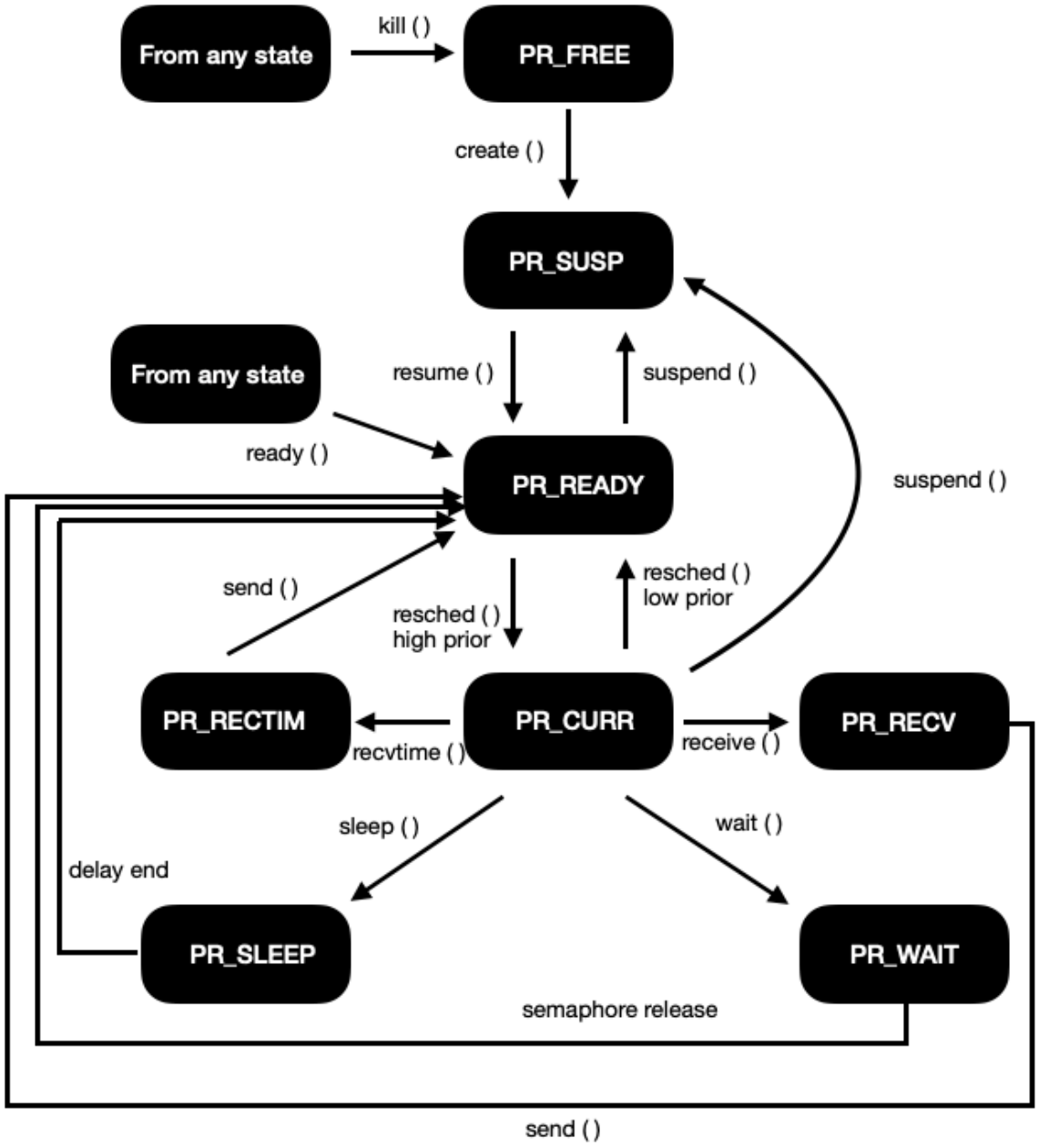
```
#define isbadpid(x) ( ((pid32)(x) < 0) || \
            ((pid32)(x) >= NPROC) || \
            (proctab[(x)].prstate == PR_FREE))
```

Q3. What is the default stack size Xinu assigns each process? Where is it defined?
Ans:

The default stack size of each process that Xinu assigns in 65536 which is in **process.h** file

```
#define INITSTK     65536   /* Initial process stack size       */
```

Q4. Draw Xinu's process state diagram.
Ans:



Q5. When is the shell process created?
Ans:

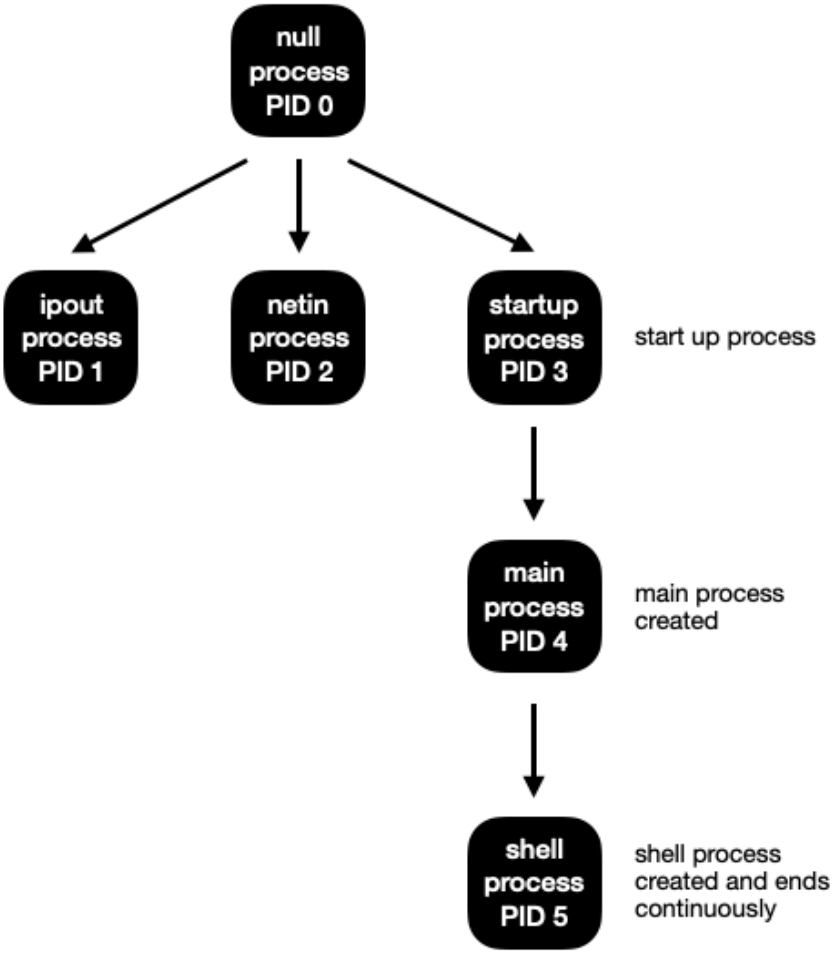
The shell process is initially created in `system/main.c` during the main process. After its initial creation, the main process enters a loop, continuously waiting for the shell process to terminate so it can recreate it and restart the cycle.

```
resume(shpid = create(shell, 8192, 50, "shell", 1, CONSOLE));
```

```
  /* Wait for shell to exit and recreate it */
  while (TRUE) {
      if (receive() == shpid) {
      sleepms(200);
      kprintf("\n\nMain process recreating shell\n\n");
      resume(shpid = create(shell, 4096, 20, "shell", 1, CONSOLE));
      }
  }
```

Q6. Draw Xinu's process tree (including the name and identifier of each process) when the initialization is Complete.

Ans:



Q7. what is the effect of the “receive()” call in the test cases provided? What would happen if that function call was not present? Hint: see receive.c and send.c files. Include the answer to this question in the report.

Ans:

The receive() function ensures that the parent process waits for its child process to complete execution. Without this function, both the parent and child processes may execute concurrently, making the execution flow unpredictable and lacking any discernible pattern.

P1: Cascading termination

include:

process.h - Added user_process flag, of type bool8, to the PCB to differentiate between user and system process.

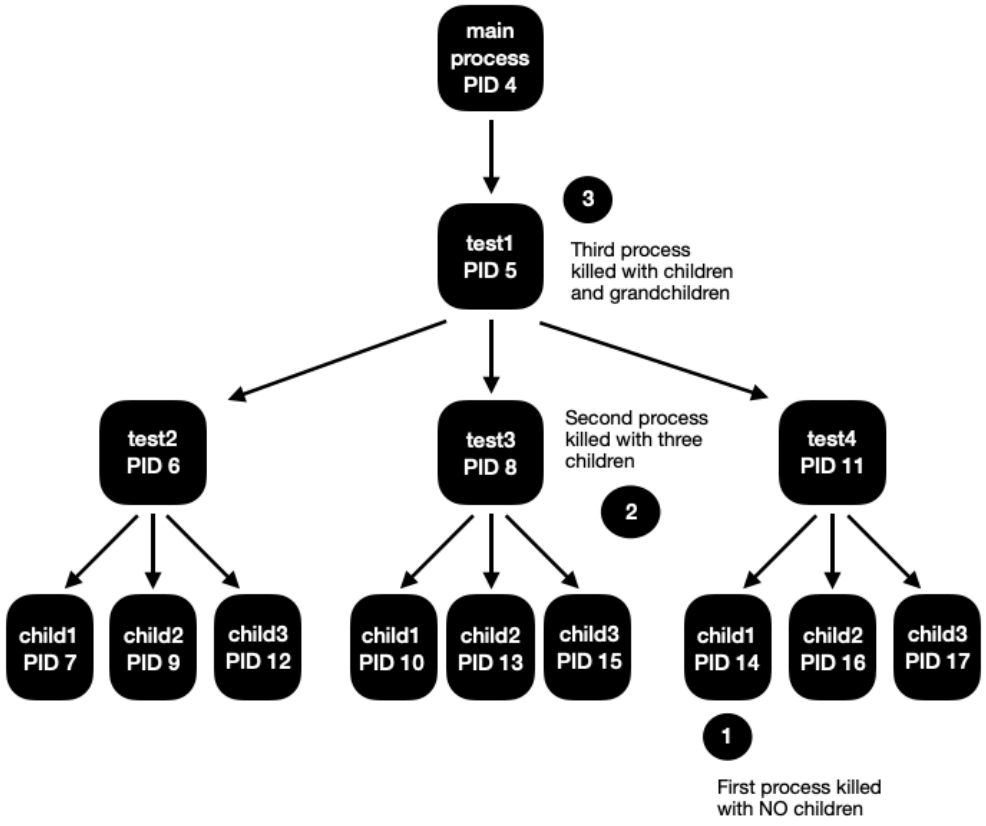
system:

create.c – Set the user_process flag to FALSE by default. Change this flag to TRUE for user processes, or keep it as FALSE for system processes, as needed, after their creation.

main.kill – Created a file to spawn 17 test cases in the form of process tree to demonstrate cascading termination and outputting the process tree before and after each termination.

kill.c – kill() function is modified to implement cascading termination of child processes. If the process to be terminated is a user process, the function checks for any child processes. If child processes are found, the kill function is recursively called on each of them.

Process tree diagram:



P2: Process creation and stack handling

include:

prototypes.h - Added extern fork () function, of return type pid32 in prototypes.h file.

system:

fork.c – To implement the fork functionality, a fork.c file was added by modifying a copy of create.c. In the fork() function, the process table entry for the child process is set up based on the parent's properties. The stack is managed by copying the parent's stack content to the child process using stack frame base pointers.