Project Report

Answers to question 1-7:

Q1. What is the maximum number of processes accepted by Xinu? Where is it defined?
Ans1: Maximum number of processes accepted by Xinu is **100**. It is defined in **/include/conf.h** file as NPROC defined.

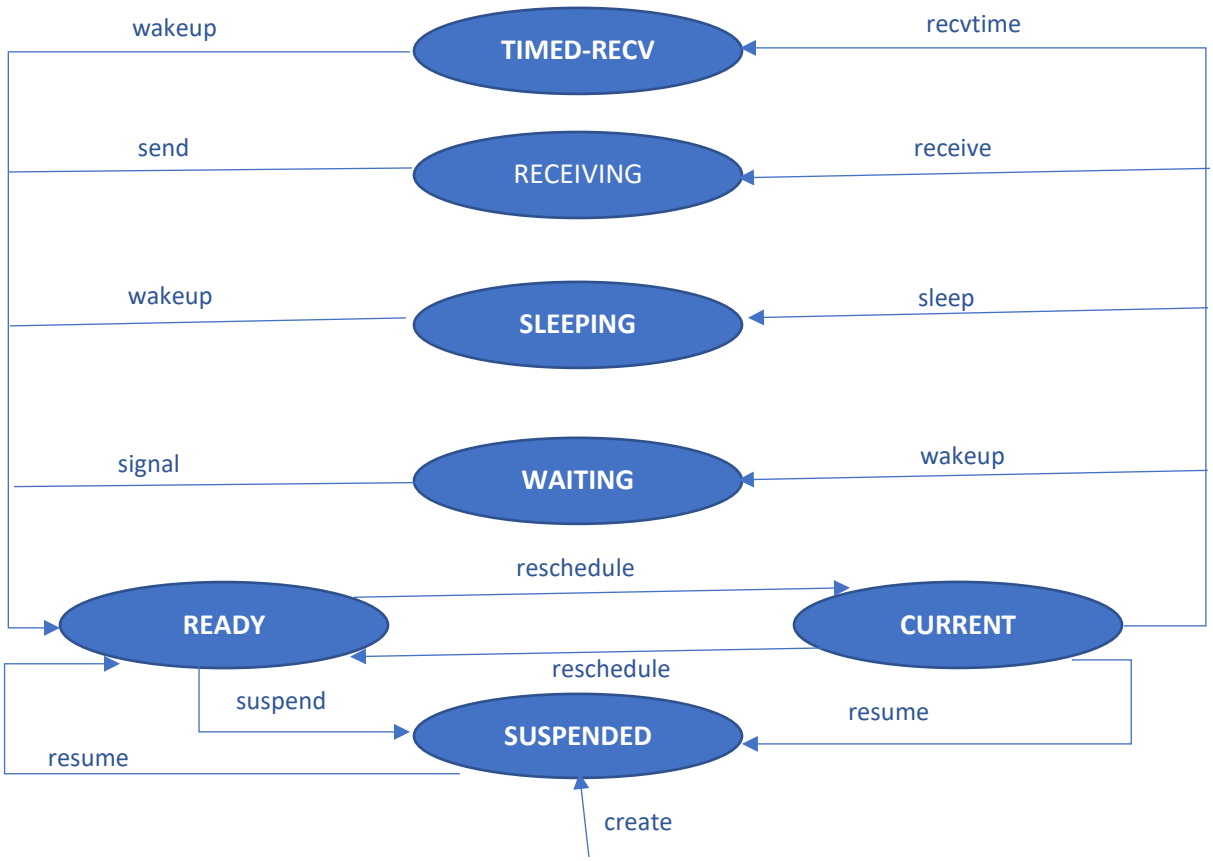
Q2. What does Xinu define as an "illegal PID"?
Ans2: Xinu defines pid as illegal or bad **if process id is less than 0 or greater than 100(Max number of processes accepted)processes or the process table entry is unused**.

Q3. What is the default stack size Xinu assigns each process? Where is it defined?
Ans3: Default stack size is **65536** for each process. It is defined **in process.h as INITSTK**.

Q4. Draw Xinu's process state diagram.
Ans4: Following are the states in the process as seen in /include/process.h

```
#define  PR_FREE0        /* Process table entry is unused     */
#define  PR_CURR         1       /* Process is currently running          */
#define  PR_READY        2       /* Process is on ready queue             */
#define  PR_RECV         3       /* Process waiting for message           */
#define  PR_SLEEP        4       /* Process is sleeping                   */
#define  PR_SUSP         5       /* Process is suspended                  */
#define  PR_WAIT         6       /* Process is on semaphore queue  */
#define  PR_RECTIM       7       /* Process is receiving with timeout     */
```

****below reference is taken from operating system design -The XINU approach



Q5. When is the shell process created?
Ans5: Shell process is created in the main function, after the null user is created and main function is called.

Q6. Draw Xinu's process tree (including the name and identifier of each process) when the initialization is complete.
Ans6:

|----- > pid- 0 Null process(prnull)
|----- > pid-1 Remode disk system initialization process(rdsproc)
|----- > pid-2 IP output process(ipout)
|----- > pid-3 Network input process(netin)
|----- > pid-4 Startup process (gets killed after spawning main)
|----- > pid-5 Main process(Main process)- this process I written by user and may have any number of functions(initialization ends here ,but continuing to explain the process in below output)
|----- > pid-6 Shell process
|----- > pid-7 ps

Output of ps in XINU:

```
Pid Name             State Prio Ppid Stack Base Stack Ptr  Stack Size Time elapsed ()
--- ---------------- ----- ---- ---- ---------- ---------- ---------- ---------------
  0 prnull           ready    0    0 0x005FDFFC 0x00FFFEB0     8192     5721
  1 rdsproc          susp   200    0 0x005FBFFC 0x005FBFC8    16384     5721
  2 ipout            wait   500    0 0x005F7FFC 0x005F7F20     8192     5720
  3 netin            wait   500    0 0x005F5FFC 0x005F5EC0     8192     5720
  5 Main process     recv    20    4 0x005E3FFC 0x005E3F54    65536     5718
  6 shell            recv    50    5 0x005F3FFC 0x005F3C7C     8192     5679
  7 ps               curr    20    9 0x005F1FFC 0x005F1DA8     8192       33
```

Question 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.

Ans7: Receive function returns "waiting" to the caller of the function and waits until the message is received.
Receive function call helps in synchronization between child and parent thread. It is a message passing mechanism. Receive() **waits until the message is received from the currently executing process**, that it is completed. 4 bytes in TCB is reserved for received message. On completion of thread, it updates the TCB. Receive monitors tcb block and ends the wait state from the parent process.
If the receive function was not present, the parent process might complete before the child is completed, leading the child to return to a non existing process- Exception.

Programming Assignment Report

Programming Assignment 1 Report:

Objective- To print the time elapsed since the start of the process, on performing a ps in the shell.

Files Touched:

Include files:

1) /include/clock.h

Variable added	Scope	file	Type of variable	Purpose/Functionality
ctr1000	Global	Clock.h	`extern uint32`	To declare it globally.

2) /include/process.h

Variable added	Scope	Function name	Type of variable	Purpose/Functionality
procent->pctr1000	Global	Process.h	struct procent	Add this to procent, as it is per process specific

System files:

1) /system/clkhandler.c -- >

Variable added	Scope	Function name	Type of variable	Purpose/Functionality
ctr1000	Global	Clkhandler()	`extern uint32`	To count the time in milli seconds. The initially present counters counted number of seconds and milliseconds separately.

2) /system/initialize.c

Variable added	Scope	Function name	Type of variable	Purpose/Functionality
Prptr->pctr1000	Global	Sysinit()	struct procent entry	Update this value with ctr1000 for null process at the start of the process, since null is not created by create function

3) /system/create.c

Variable added	Scope	Function name	Type of variable	Purpose/Functionality
Prptr->pctr1000	Global	create(…)	struct procent entry	Updated with ctr1000 for every new process that gets created

Shell files:

1) /shell/xsh_ps.c

Variable added	Scope	Function name	Type of variable	Purpose/Functionality
`timeNow`	Local	xsh_ps(…)	`uint32`	Updated with ctr1000 for the current time when ps is called

Also added a print column entry of time elapsed in ms with diff between time that the process was created and the current time. –(`timeNow-(prptr->prctr1000))`)

Programming Assignment 2:

Objective- to implement fork functionality.

Implementation logic:

To implement fork, 1st the child stack is created using the same procent table as parent, then the child stack is filled carefully with the contents taken from parent stack, only till the frame before fork. Note, all the function pointers are offsetted with respect to the child, by calculating the offsets using function pointer traversal from parent. The stack is filled from bottom to top. Finally, the pointer is got back to bottom, and all 8 registers along with the interrup register is updated and eax register is filled with the return value of child – NPROC , the ebx is set to the function pointer of the frame before fork, where the child should return on context switch.

Variable list –

```
/* Declaring variables to use in code */
        uint32          ssize;          /* Stack size in bytes              */
        pri16   priority;/* Process priority > 0          */
        char            *name;          /* Name (for debugging)             */
        uint32          savsp, *pushsp; /* to save the stack pointers of the child*/
        intmask         mask;           /* Interrupt mask                   */
        pid32           pid;            /* Stores new process id            */
        struct  procent*prptr;          /* Pointer to proc. table entry */
        struct  procent *currptr;       /*parent child proc. table entry */
        int32           i,j;            /* for loop counter */
        uint32          *a;             /* Points to list of args   */
        uint32          *saddr;         /* Stack address                    */
        uint32          *psaddr;        /* parent stack address             */
        uint32          total_offset;   /*offset of stackptr from the parent basp ptr*/
        unsigned long   *sp, *fp;       /* pointers for parent */
        uint32 offset,base_offset;      /*offsets to create child stack */
        unsigned long *fp_copy,addr_fp_copy;  /*copy of fp and address of fp */
        unsigned long *saddr_pre_copy;                  /*coppy of saddr with value of stack base
pointer */
```

Files touched:

1) /system/fork.c
2) /system/main.c

Programming assignment 3:

Objective: Tracking of system calls-Frequency and avg clock cycles of create, kill, ready, sleepms,suspend, wait, wakeup, yield.

Implementation:

Changes in create, kill, ready, sleepms,suspend, wait, wakeup, yield

This is implemented by adding the assembly counter code from the manual in every function mentioned above. To calculate frequency of every function call, frequency with respect to every process is incremented by one, every time the process is called.

To calculate the average clock cycles, every time the function is called, clock count is started and after completion ended. The difference of start and stop is added to the clock cycles w.r.t every process.

Implementation in pr_status_syscall_summary()

The total of clock cycle is finally divided by frequency in the `pr_status_syscall_summary()`

This function is used to print the table with the details of the summary with respect to each process and each syscall. The total frequency and average clock cycles is displayed in table format.

*****Frequency and clockCycles are added to struct procent**

Variables ;

```
struct procent *prptr;
uint32 i,k;
bool flag=FALSE;
char arr[][40]={"create","kill","ready","sleepms","suspend","wait","wakeup","yield"};
```

Files touched:

System:

1) /system/create.c
2) /system/kill.c
3) /system/ready.c
4) /system/sleep.c

5) /system/suspend.c
6) /system/wait.c
7) /system/wakeup.c
8) /system/yield.c
9) /system/pr_status_syscall_summary.c
10) /system/main.c

Include:

1) /include/process.h
2) /include/xinu.h –included –(`stdbool.h, stdint.h`)

********outputs to the codes in / output_of_p1_p2_p3_and_main_fies folder.