Sistemas Operativos

TP2: Construcción del Núcleo de un Sistema Operativo

# Integrantes:

## Nicolás Kuyumciyan 55165

## Magdalena Vega 55206

## Lucas Casagrande 55302

Aplicaciones

Previas de Arquitecturas:

Clear - Borra toda la pantalla.

Time - Muestra la hora y la fecha

Help - Muestra todos los comando disponibles

Change time - Te permite cambiar el =empo y la fecha

Whoami - Te dice quien sos.

Keyboard - Muestra gráficamente la distribución del teclado

Colors - Permite cambiar los colores de la consola.

Screen time - Permite cambiar el =empo de inac=vidad hasta que aparezca el salva pantallas

Piano - Le permite al usuario generar sonido en forma de un piano

Beep - Genera un sonido predefinido

Nuevas Aplicaciones:

IPCS – Muestra todos los IPC’s existentes

PS – Muestra todos los procesos

GAME – Ejecuta el juego

DRAW – Como un paint

Juego

IDEA:

La idea del juego

DIVISION:

System Calls

Previas de Arquitecturas:

write

erase\_scr

get\_str

get\_char

rtc\_read

rtc\_write

colors

scr\_time

beep

piano

songs

nuevas system calls:

memory management sys calls

malloc

free

ipc sys calls

mkfifo

openfifo

closefifo

writefifo

readfifo

readfifobloq

showipcs

graphic sys calls

set\_screensaver

colors\_graphic

clear

draw\_circle

draw\_image

draw\_text

enter\_draw\_mode

exit\_draw\_mode

draw\_erasable\_circle

undraw\_erasable\_circle

get\_char\_from\_buffer

set\_event\_keyup

unset\_event\_keyup

set\_event\_keydown

unsert\_event\_keydown

multitasking sys calls

new\_process

end\_process

get\_all\_process

sleep

Funcionamiento

IPC:

Se decidió implementar named pipes (fifo) para la comunicación entre procesos. Esto fue por la simplicidad y efectividad que provee ante los objetivos propuestos.

Dado que el trabajo no tiene implementado un file system los pipes son de tamaño fijo (4KB), son alocados en tiempo de creación y liberados al cerrarlos.

Para lograrlo se implementó un array de estructuras que contienen el file descriptor asociado con el pipe, su nombre y un puntero a la dirección de memoria donde se encuentran los datos.

Para mayor versatilidad se implementaron dos tipos diferentes de read: bloqueante y no bloqueante. El bloqueante es importante para los procesos que no tienen nada que hacer hasta que puedan leer algo, lo cual permite marcarlos como inactivos y no consumen tiempo de procesador. Una vez que se escribe en el pipe donde se quedaron bloqueados se los marca nuevamente como activos y continuan con su ejecución.

Scheduler:

EL scheduler utiliza un algoritmo sencillo de round-robin.

además los procesos pueden estar en estado ready (listos para ejecutarse) o waiting (que están esperando alguna acción especial para desbloquearse).

Adicionalmente hay un proceso inactivo del sistema que se ejecuta cuando todos los otros procesos están inactivos. Dicho proceso es un hlt que evita que se use el CPU cuando no es necesario.

Los procesos guardan la siguiente información: su nombre, estado, id y el id del proceso que lo creó, además de todos los datos necesarios para realizar el contex switch y si el proceso tiene el foreground o no.

Para crear un proceso simplemente se necesita llamar a la system call correspondiente pasándole el nombre y el puntero a función donde el programa va a ejecutarse.

Los procesos se pueden bloquear mediante un sleep por una cantidad determinada de tiempo, además de bloquearse cuando se pide un recurso que no está disponible (por ejemplo cuando la consola pide el texto ingresado por el usuario, pero el usuario no ingreso nada todavía).

Memory Management:

decidimos para que sea más sencillo que al dar memoria al usuario se le van a dar múltiplos de 4kb.

Para ello se implementó un vector con la cantidad de posiciones como bloques de 4kb haya, y luego de ese vector, hay otro igual donde por cada dirección dice cuantos bloques se dieron (esto es para poder saber cuantos se tienen que liberar). cuando un usuario pide memoria, se marca las posiciones que se llevó como “usadas” y se le da la dirección de donde empieza el primero.

para el free lo que se hace es deducir a partir de la dirección que se pasa por parámetro qué bloque es y se marca ese bloque como “libre”.

se implementó una estructura para poder hacer esto que contiene el vector, la cantidad de bloques que va a haber y un int que dice cual fue el último bloque que se alocó (Esto se hace para no tener que recorrer todo el vector, asumiendo que nada de lo anterior se liberó)