pandOS

The PandOS operating system is an educational project consisting in the implementation of a kernel/OS designed to run on µMPS. This documentation describes the implementation for the first phase of the project.

DESIGN CHOICES

Platform

The pandOS repository has been deployed on GitHub to help the collaboration among the authors, combined with GitFlow to manage branches.

Building

CMake was adopted to automate the building process for the generation of the makefile.

Documentation

The guideline to write the documentation is the Doxygen standard, used to have a consistent way to comment the functions.

MODULES

PANDOS_CONST

This header file contains utility constants & macro definitions. In addition to the pre-existing ones, this constants have been declared:

MAXPROC
Max number of concurrent processes pandOS can support.

MININT
Identifier with the lowest value, used for the first dummy semaphore at the start of the ASL.

MAXINT
Identifier with the highest value, used for the second dummy semaphore at the end of the ASL.

MAXSEM
Total number of semaphores to be inserted in the ASL, counting also the 2 dummies ones.

PANDOS_TYPES

This header file contains utility types definitions. It defines:

typedef signed int cpu_t

typedef unsigned int memaddr

typedef struct context_t

It contains the following members:

unsigned int c_stackPtr;

unsigned int c_status;

unsigned int c_pc;

typedef struct support_t

It contains the following members:

int sup_asid;	process ID
state_t sup_exceptState[2]	old state exceptions
context_t sup_exceptContext[2]	new contexts for passing up

typedef struct pcb_t

Process Control Blocks (pcbs). It contains the following members:

struct pcb_t *p_next	ptr to next entry
struct pcb_t *p_prev	ptr to previous entry
struct pcb_t *p_prnt	ptr to parent
struct pcb_t *p_child	ptr to 1st child
struct pcb_t *p_next_sib	ptr to next sibling
struct pcb_t *p_prev_sib	ptr to prev. sibling
state_t p_s	processor state
cpu_t p_time	cpu time used by proc
int *p_semAdd	ptr to semaphore on which proc is blocked

The children list of a pcb is double linked but not circular.

typedef struct semd_t

Active Semaphore List (ASL). It contains the following members:

struct semd_t *s_next	ptr to next element on queue
int *s_semAdd	ptr to the semaphore
pcb_PTR s_procQ	ptr to tail of the queue of procs. blocked on this sem.

PCB

HIDDEN pcb_t *pcbFree_h NULL-terminated single, linearly linked list containg the unused PCBs

Since the pcb in the free list are all equals, the list is considered as a stack.

Process Control Blocks functions

HIDDEN pcb_t *resetPcb(pcb_tp)

Resets all the values of a pcb pointer to NULL.
Parameters
p The pointer to the PCB that has to be resetted.
Returns
The pointer to the pcb.

void initPcbs()

Initializes the pcbFree list. This function should be called only once during initialization.

void freePcb(pcb_t *p)

Deallocates the element pointed by p.
Parameters
p Pointer to the pcb that has to be inserted in the pcbFree list.

pcb_t *allocPcb()

Allocates a PCB and provides initial values for all of his camps.
Returns
NULL if the pcbFree list is empty otherwise a pointer to the removed pcb.

void initPcbs()
Initializes the pcbFree list. This function should be called only once during initialization.

pcb_t *mkEmptyProcQ()

Initializes a new empty process queue.
Returns
A tail pointer to an empty process queue.

int emptyProcQ(pcb_t *tp)

Checks if the queue pointed by tp is empty.
Parameters
tp Tail pointer of the queue.
Returns
TRUE if the queue is empty, FALSE otherwise.

void insertProcQ(pcb_t **tp, pcb_t *p)

Inserts the pcb pointed by p into the queue pointed by tp.
Parameters
tp Tail pointer of the queue.
p Pointer to the pcb.

pcb_t *headProcQ(pcb_t *tp)

Returns the pointer to the head of the tp process queue, without removing it.
Parameters
tp The pointer to the tail of the process queue.
Returns
The pointer to the head of the process queue, NULL if the queue is empty.

pcb_t *removeProcQ(pcb_t **tp)

Removes the oldest element (the head) from the tp queue.
Parameters
tp The pointer to the queue.
Returns
The pointer to the element removed from the list, NULL if the queue is empty.

pcb_t *outProcQ(pcb_t **tp, pcb_t *p)

Removes the PCB pointed by P from the process queue pointed by tp.
Parameters
tp The pointer to the queue.
p The pointer to the PCB that has to be removed.
Returns
The pointer to the removed PCB, NULL if the PCB pointed by p is not in the queue.

Definitions of Process Tree functions

HIDDEN pcb_t *trim(pcb_t *p)

This funcion takes as input a pointer to a PCB who has to be removed from his tree.
Parameters
p The pcb pointer that has to be removed from his tree.
Returns
The pointer to the PCB whose fields have been set to NULL.

int emptyChild(pcb_t *p)

Inspects if the PCB pointed by p has a child.
Parameters
p The pointer to the PCB that has to be inspected.
Returns
TRUE if the PCB pointed by p has no children, FALSE otherwise.

void insertChild(pcb_t *prnt, pcb_t *p)

Inserts the PCB pointed by p as a child of the PCB pointed by prnt.
Parameters
prnt The pointer to the PCB which will become parent of p.
p The pointer to the PCB which will become child of prnt.

pcb_t *removeChild(pcb_t *p)

Removes the first child of the PCB pointed by p.
Parameters
p The pointer to the PCB whose first child will be removed.
Returns
The pointer to the first child of the PCB, NULL if the PCB doesn't have a child.

pcb_t *outChild(pcb_t *p)

Removes the PCB pointed by p from the list of his parent's children.
Parameters
p The pointer to the PCB that will be removed.

Returns
The pointer to the PCB, NULL if the PCB doesn't have a parent.

ASL

Active Semaphore List functions. It defines:

HIDDEN semd_t* semdFree_h	NULL-terminated single, linearly linked unused semaphore list
HIDDEN semd_t* semd_h	NULL-terminated single, linearly linked active semaphore list

HIDDEN semd_t *findPrevSem(int *semAdd)

This function takes as input a semAdd and returns the last semaphore in semd_h whose identifier is lower than the one passed as argument.
Parameters
semAdd Semaphore identifier.
Returns
The last semaphore whose semaphore is lower than semAdd.

int insertBlocked(int *semAdd,pcb_t *p)

Insert the pcb pointed to by p at the tail of the process queue associated with the semaphore whose physical address is semAdd and set the semaphore address of p to semAdd.
Parameters
semAdd Semaphore identifier.
p Pointer to the PCB to be inserted.
Returns
TRUE if a new semaphore descriptor needs to be allocated, FALSE otherwise.

pcb_t *removeBlocked(int *semAdd)

Search for a semaphore whose descriptor is semADD. Remove the first pcb from its process queue and return apointer to it.
Parameters
semAdd Semaphore identifier.
Returns
The pointer to the head from the process queue associated with the semaphore descriptor.

pcb_t *outBlocked(pcb_t *p)

Remove the pcb pointed to by p from the process queue associated with p's semaphore.
Parameters
p Pointer to the pcb to be removed.
Returns
A pointer to the removed PCB. Returns NULL if p does not appear in the process queue.

pcb_t *headBlocked(int *semAdd)

The a pointer to the head of the process queue associated with the semaphore semAdd.
Parameters
semAdd Semaphore identifier.
Returns
The first element of the process queue associated with the semaphore semAdd or NULL if semAdd is not found.

void initASL()

Initialize the semdFree list, this method will be only called once during data structure initialization.

MEMORY

It defines the following function:

void *memcpy(void *dest, const void *src, size_t n)

Copies bytes from an address to another one.

Parameters

dest Destination.

src Source.

len Length of the bytes to be copied.

Returns
A pointer to the destination address.

INITIAL

Entry point of pandos project. Setups the nucleus. It contains:

Variables

unsigned int processCount

Counts active processes.

unsigned int softBlockCount

Counts blocked processes on device semaphores.

pcb_t *readyQueue

Queue of the processes in the running state.

pcb_t *currentProcess

The active process.

SEMAPHORE semaphoreList[DEVICE_NUMBER]

Device semaphores.

SEMAPHORE swiSemaphore

Semaphore for the System Wide Interval Timer.

Function

int main(void);

PandOS entry point.

Returns
The exit code.

SCHEDULER

void scheduler()

Picks the first process from the ready queue and and executes it. Before of doing so inserts current one back in the ready queue.

cpu_t getTimeSlice()

Gets the time elapsed as current process.

Returns
The difference between the moment the current process has been selected and the current time.

EXCEPTIONS

void TLBExcHandler()

Handles a TLB exception.

void generalTrapHandler()

Handles a Program Trap.

void exceptionHandler()

Handles exceptions passing them to their custom handler.

SYSCALLS

void sysHandler()

Handler for the syscalls. It gets called by the exception handler.

void createProcess(state_t * statep, support_t * supportp)

SYS1: creates a new process with the state and the support structure passed as parameters.

Parameters

statep State of the new process.

supportp Support structure of the new process.

Returns
An exit code who specifies if the operation was completed successfully.

void terminateProcess()

SYS2: terminates the running process and all its progeny recursively.

void passeren(int *semAdd);

SYS3 (P): Does a P operation on the semaphore passed as parameter.

Parameters
semAdd Pointer to the semaphore to perform the P on.

pcb_t* verhogen(int *semAdd);

SYS4 (V): Does a V operation on the semaphore passed as parameter.

Parameters
semAdd Pointer to the semaphore to perform the V on.

Returns
The pointer to the PCB that was unblocked from the V, otherwise returns NULL.

void waitIO(int intlNo, int dNum, bool waitForTermRead);

SYS5: waits for an I/O operation. It blocks the current process on a (sub)device semaphore specified by the parameteres.

Parameters

intlNo Interrupt line.

dNum Device number of that line.

waitForTermRead Specifies if the terminal reads or writes.

void getCpuTime()

SYS6: returns the total time a process has been active, storing the value in the v0 register.

void waitForClock()

SYS7: blocks the current process in the System wide interval semaphore until the next SW interrupt.

void getSupportStruct()

SYS8: returns the pointer to the currentProcess' support struct, saving it in the register v0.

INTERRUPTS

define CAUSE_IP_GET(cause,line) (cause & CAUSE_IP_MASK) & CAUSE_IP(line)

A macro to get the line cause of an interrupt.

void interruptHandler()

Brief Handler for the interrupts. It gets called by the exception handler.

INITPROC

void test()

First function that will be called by the phase 2

SYSSUPPORT

void generalExceptionHandler()

Handles the third level exceptions

void syscallExceptionHandler(int *sysNumber*, support_t **support*);

Handles the syscalls not handled by the level 2

Parameters

sysNumber Number of the syscall who generated the exception

support Pointer at the support structure of the process that caused the exception

void programTrapExceptionHandler(support_t **support*)

Trap Exceptions Handler

Parameters

support Pointer at the support structure of the process that caused the exception

void terminate(support_t **support*)

Terminate a process, wrapper of the level 2 function with the same goal

Parameters

c*urrentSupport Pointer to the support structure of the current process.*

void getTOD(support_t **support*)

Stores the TOD in the current process v0 register

void writePrinter(char* *string*, int *len*, support_t* *support*)

Writes a string to the printer

Parameters

string Pointer to the first character of the string.

len Lenght of the string.

support Pointer to the support structure of the current process.

void writeTerminal(char **string*, int *len*, support_t* *support*)

Writes a string to the terminal

Parameters

string Pointer to the first character of the string.

len Lenght of the string.

support Pointer to the support structure of the current process.

void readTerminal(char *_string_, support_t *_support_)

Reads a string from the terminal used by the current process

Parameters

string Pointer to the first character that will store the string.

support Pointer to the support structure of the current process.

VMSUPPORT

void initSwapStructs()

Initializes third level structs.

void clearSwap(int _asid_);

Clears the swap table

int replacementAlgorithm();

Selects a new frame where to write

void updateTLB(pteEntry_t *_newEntry_)

Updates the TLB

Parameter

newEntry Pointer to the new entry in TLB

void executeFlashAction(int _deviceNumber_, unsigned int _pageIndex_, unsigned int _command_, support_t *_support_)

Reads or writes a flash device

Parameters

deviceNumber Device index.

primaryPage Page index in primary memory.

command Command to be used.

currentSupport Pointer to the support structure of the current process.

void pager()

Handles TLB Page Fault exceptions

void uTLB_RefillHandler()

Handles TLB Refill exceptions