Document of FlitOS

—How does Flappy Bird soar into the sky? Try FlitOS!

Professor: **Christopher Nitta**

Working Group: **Yifeng Shi**
Xiaoxing Chen
Siyuan Liu
Tianxiao Cheng

ECS251 Operating System
Project Final Phase

UCDAVIS
UNIVERSITY OF CALIFORNIA

Contents

0 Source Code 2

1 Introduction 2

2 Highlight Features 2

2.1 C++ Based Kernel 2

2.2 QEMU Simulation Available 3

2.3 Gaming Graphics 3

3 API 4

3.1 Thread Management 4

3.2 Event Management 4

3.3 Memory Management 5

3.4 Graphics API 5

3.5 Thread Synchronization 8

3.6 Inter-thread Communication 9

3.7 Extra System Calls 10

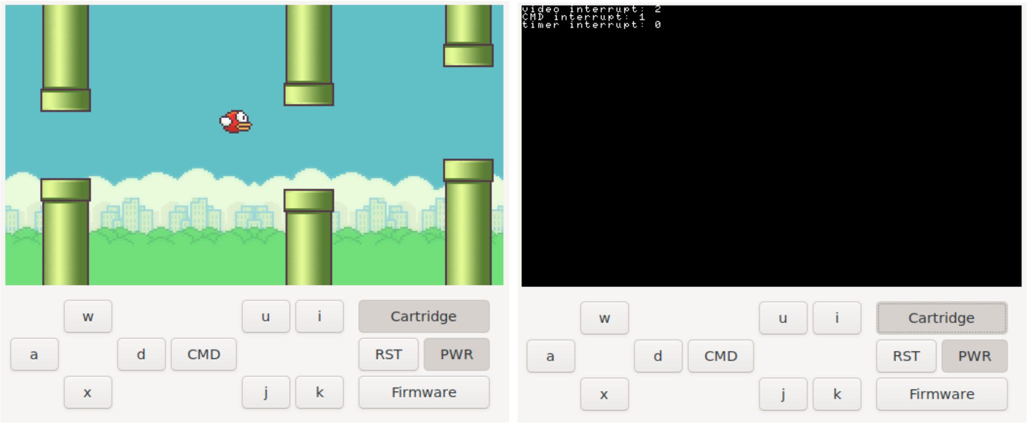
0 Source Code

- FlitOS: our OS
- Team Porting Choice: with Group2
- Instructor Porting Choice: with Group5

1 Introduction

FlitOS (yi**F**engshi siyuan**L**iu x**I**aoxingchen **T**ianxiaocheng OS), is a real-time operating system kernel that developed for the RISC-V based game console simulator. 'flit' indicates 'to fly quickly', which means our FlitOS is fast, simple and reliable. Detailed descriptions of the APIs are provided for the developers who are interested in developing games on the OS.

Our primary goal we want the OS to achieve is to provide the game developers with a software layer in between the simulated game console and their game application [2]. Our design is highly inspired by Linux operating system. Ideally, this layer should manage the hardware resources, including the CPU, RAM and I/O devices, so that the developers can focus on the higher-level logic of game development without worrying about directly manipulating the hardware. Our APIs is still under active design and development by our team. Currently APIs cover thread management, event management, memory allocation, thread synchronization and sprite operation.



```
video interrupt: 2
CMD interrupt: 1
timer interrupt: 0
```

(a) Preview of Graphics Mode

(b) Preview of Text Mode

Figure 1: Preview

2 Highlight Features

2.1 C++ Based Kernel

In order to make the code reusable, efficient and decoupled, we decided to use C++ as main language for kernel development. Usually C++ is not recommended in bare-metal environment because of the limited RAM and FLASH space, but those are not concerns in our riscv-console simulator.

However, C++ STL are highly dependent on the implementation of the OS. For example, `std::shard_ptr` is thread safe on Linux and Windows, but the module does not even know the existence of FlitOS. No thread safety will be guaranteed. Similarly, `std::mutex`, `std::condition_varible` are not available in bare-metal environment.

Therefore, in FlitOS kernel, we took advantage of C++ grammar but not C++ STL. We have our own implementation of STL containers like `ecs::vector`,`ecs::deque`,`ecs::list`. Those containers can take either raw memory allocator or thread-safe `ecs::allocator` for memory allocation.

2.2 QEMU Simulation Available

During our development, we found the debug environment of riscv-console is not that friendly for developer. For example, it is hard to inspect the value of a specific variable while debugging. The second problem is the disassembly window is not compatible with C++ template. The third problem is the logging. We need some log information to narrow down the problem. But printing log information in text-mode window is a bit inefficient.

To overcome the problems above, we set up another development environment: QEMU + gdb + gdbgui. We found a RISC-V QEMU docker on github. The `virt` machine has a UART port and is a perfect interface for log output. However the docker environment does not support graphic GUI. Therefore all kernel related components like scheduler, synchronization objects are tested on QEMU while graphics related APIs are tested on riscv-console.

2.3 Gaming Graphics

In our cartridge, we have implemented periodic moving columns for our program to achieve the effect of continuous flight of our bird sprite, as well as periodically natural falling of the bird sprite to mimic the gravity effect.

Based on our group's firmware, we have implemented a series of functions. For example, Multi-thread Scheduling, Locks, Callback functions(that is, the Up-call for the cartridge), and data structures, like lists, etc. In this full-feature version, we:

- open thread 1 to implement the response to the control buttons and to safely modify the bird coordinate information by obtaining a mutually exclusive lock.

- open thread 2 to use periodic interrupts to trigger a thread switch to achieve the natural fall of the bird, and to obtain a mutually exclusive lock to safely modify it with thread 1.

- open thread 3 to calculate the collision between the bird and the column.

  If a collision is detected, all threads will stop running via judging a semaphore shared between all threads, and a warning message presented via a series of sprites will be shown in the screen and waiting for user to restart the game.

3 API

3.1 Thread Management

NAME	`threadCreate`
DESCRIPTION	starts a new thread in the calling thread
SIGNATURE	`uint32_t threadCreate(void (*f)(void*), void* arg)`
PARAMETERS	`f` - the function that the new thread invokes `arg` - arguments of `f`
RETURN VALUE	0 or positive number - thread id -1 if failed

NAME	`threadJoin`
DESCRIPTION	wait for the specified thread to be terminated and join with the current thread
SIGNATURE	`int threadJoin(int thread_id)`
PARAMETERS	`thread_id` - target thread ID
RETURN VALUE	0 if success -1 if failed

NAME	`threadYield`
DESCRIPTION	yield current thread
SIGNATURE	`int threadYield()`
PARAMETERS	no parameters required, yield current thread.
RETURN VALUE	0 if success -1 if failed

NAME	`threadSleep`
DESCRIPTION	sleep current thread for miliseconds
SIGNATURE	`int threadSleep(int n_100ms)`
PARAMETERS	sleep for a specific time period.
RETURN VALUE	0 if success -1 if failed

3.2 Event Management

NAME	`register a event handler`
DESCRIPTION	when an event occurs, the OS will open a separate thread to call the callback function to prevent the OS interrupt handling thread from being blocked by the callback function, ensuring that the OS can respond quickly to a series of consecutive events.

SIGNATURE	`int registerEvent`
	`(int event_type, void (*f)(int))`

PARAMETERS	`event_type`: `1` - button event `2` - cartridge event `f` - handler, customized function handling registered event.
RETURN VALUE	0 if success -1 if failed

NAME	`deregister a event handler`
DESCRIPTION	remove event handler
SIGNATURE	`void deregisterEvent(int handler_descriptor);`
PARAMETERS	`eventType`, which event is going to register handler_descriptor, globally unique mark for each handler.
RETURN VALUE	void

3.3 Memory Management

NAME	`malloc`
DESCRIPTION	Allocate the memory. This method is thread safe.
SIGNATURE	`void* malloc(int size);`
PARAMETERS	`size` - size of memory the user wants to allocate
RETURN VALUE	address of the memory. `NULL` if allocation failed.

NAME	`free`
DESCRIPTION	Deallocate the memory. This method is thread safe.
SIGNATURE	`void free(void* p);`
PARAMETERS	`p` - address of the memory
RETURN VALUE	`void`

3.4 Graphics API

NAME	`linePrintf`
DESCRIPTION	writes the string pointed by **format** to the text mode at the line specified by **line__idx**. If **format** includes format specifiers (subsequences beginning with %), the additional arguments following **format** are formatted and inserted in the resulting string replacing their respective specifiers.
SIGNATURE	`int linePrintf(uint32_t line_idx, const char *format, ...)`
PARAMETERS	`int32_t line_idx` - index of line in text mode `const char *format` - string that contains the text to be written `...` - additional arguments
RETURN VALUE	0 if success -1 if failed

NAME	`setLargeSpriteControl`
DESCRIPTION	set parameter to large sprite control register
SIGNATURE	`void setLargeSpriteControl(uint32_t idx, uint32_t h, uint32_t w, uint32_t x, uint16_t y, uint32_t palette)`
PARAMETERS	`idx` - index address identifier of largeSprite, range of values from 0 to 63 `h` - height of the largeSprite `w` - width of the largeSprite `x,y` - respective positions on the x and y axes `palette` - index of the palette, range of values from 0 to 3
RETURN VALUE	0 if success -1 if failed

NAME	`setBackgroundControl`
DESCRIPTION	set parameter to background control register
SIGNATURE	`void setBackgroundControl(uint32_t idx, uint32_t x, uint32_t y, uint32_t z, uint32_t palette)`
PARAMETERS	`idx` - index of the background control, value ranges from 0 to 4 `x,y` - respective positions on the x and y axes `z` - A 3-bit Z position specifies the Z plane in which the image will be rendered `palette` - index of the palette, value ranges from 0 to 3
RETURN VALUE	0 if success -1 if failed

NAME	`initBackgroundPalett`
DESCRIPTION	initialize background palette contents
SIGNATURE	`int initBackgroundPalette(uint32_t idx, uint8_t * addr, uint32_t mem_len)`
PARAMETERS	`idx` - index of the background palette, value ranges from 0 to 3 `addr - an array of uint8_t of length 4*16` `mem_len`-a constant value of 4*16
RETURN VALUE	0 if success -1 if failed

NAME	`initSpritePalette`
DESCRIPTION	initialize sprite palette contents
SIGNATURE	`int initSpritePalette(uint32_t idx, uint8_t * addr, uint32_t mem_len)`
PARAMETERS	`idx` - index of the sprite palette, value ranges from 0 to 3 `addr - An array of uint8_t of length 4*8` `mem_len`-a constant value of 4*16
RETURN VALUE	0 if success -1 if failed

NAME	`initTransparentSpritePalette`
DESCRIPTION	set the sprite palette, which is indicated by the parameter idx, to transparent.
SIGNATURE	`int initTransparentSpritePalette(uint32_t idx)`
PARAMETERS	`idx` - index of the sprite palatte, value ranges from 0 to 3
RETURN VALUE	0 if success -1 if failed

NAME	`setBackgroundDataImage`
DESCRIPTION	write data to background data image
SIGNATURE	`int setBackgroundDataImage(uint32_t idx, uint8_t * addr)`
PARAMETERS	`idx` - index of the target background data image, value ranges from 0 to 4 `addr-an array of unit8_int of length 288*512`
RETURN VALUE	0 if success -1 if failed

NAME	setLargeSpriteDataImage
DESCRIPTION	write data to large sprite data image
SIGNATURE	int setLargeSpriteDataImage(uint32_t idx, uint8_t * addr)
PARAMETERS	idx - index of the sprite palatte, value ranges from 0 to 3 addr - an array of uint8_t of length 64*64
RETURN VALUE	0 if success -1 if failed

NAME	setDisplayMode
DESCRIPTION	any illegal value that neither 1 or 0, will be interpreted with the last bit
SIGNATURE	void setDisplayMode(uint32_t mode)
PARAMETERS	mode - 0 represents the text mode, while 1 represents the graphics mode
RETURN VALUE	0 if success -1 if failed

3.5 Thread Synchronization

NAME	mutexInit
DESCRIPTION	initialize a mutex
SIGNATURE	int mutexInit()
PARAMETERS	
RETURN VALUE	a mutex descriptor l with be returned. There is an error if l <= 0.

NAME	mutexDestroy
DESCRIPTION	destroy a mutex
SIGNATURE	int mutexDestroy(int l)
PARAMETERS	l - mutex descriptor
RETURN VALUE	0 if success -1 if failed

NAME	mutexLock[1]
DESCRIPTION	lock a mutex
SIGNATURE	void mutexLock(int l)
PARAMETERS	l - mutex descriptor
RETURN VALUE	void. function will block if lock is acquired by other threads

NAME	mutexUnlock
DESCRIPTION	unlock the mutex
SIGNATURE	void mutexUnlock(int l)
PARAMETERS	l - mutex descriptor
RETURN VALUE	void. release the lock and other threads can access

NAME	`condInit`
DESCRIPTION	create a condition variable
SIGNATURE	`int condInit()`
PARAMETERS	none
RETURN VALUE	`int`. Condition variable descriptor

NAME	`condDestroy`
DESCRIPTION	destroy a condition variable
SIGNATURE	`int condDestroy(int fd)`
PARAMETERS	`int`. Condition variable descriptor
RETURN VALUE	0 if success -1 if failed

NAME	`CondSignal`
DESCRIPTION	signal a condition variable
SIGNATURE	`int CondSignal(int fd)`
PARAMETERS	`int`. Condition variable descriptor
RETURN VALUE	0 if success -1 if failed

NAME	`CondBroadcast`
DESCRIPTION	signal all threads that waiting on the condition variable
SIGNATURE	`int CondBroadcast(int fd)`
PARAMETERS	`int`. Condition variable descriptor
RETURN VALUE	0 if success -1 if failed

NAME	`CondWait`
DESCRIPTION	let the thread wait on a condition variable
SIGNATURE	`int CondWait(int cond_fd, ind mtx_fd)`
PARAMETERS	`int`. Condition variable descriptor `int`. Mutex descriptor
RETURN VALUE	0 if success -1 if failed

3.6 Inter-thread Communication

NAME	`pipeOpen`
DESCRIPTION	Create a new pipe
SIGNATURE	`int pipeOpen()`
PARAMETERS	None
RETURN VALUE	`int` pipe descriptor

NAME	pipeClose
DESCRIPTION	Close a pipe.
SIGNATURE	`int pipeClose(ind pipe_fd)`
PARAMETERS	`int` the pipe to close
RETURN VALUE	`int` pipe descriptor

NAME	pipeRead
DESCRIPTION	Read data from a pipe into a buffer. If the pipe is empty, the thread will block.
SIGNATURE	`int pipeRead(ind pipe_fd, uint8_t* buff, int len)`
PARAMETERS	`int` the pipe to read `uint8_t*` buffer for output data `int` read data length
RETURN VALUE	`int` actual data length that read into buffer

NAME	pipeWrite
SIGNATURE	`int pipeWrite(ind pipe_fd, uint8_t* buff, int len)`
PARAMETERS	`int` the pipe to write `uint8_t*` buffer for input data `int` write data length
RETURN VALUE	`int` actual data length that write to buffer

3.7 Extra System Calls

NAME	writeTargetMem
DESCRIPTION	write data to target addree of a memory handle

SIGNATURE	`void writeTargetMem(`
	`uint32_t reg_id, uint32_t src_addr, uint32_t mem_len)`

PARAMETERS	`reg_id` - id of the register that to be modified. `src_addr` - source address to copy. `mem_len` - memory length.
RETURN VALUE	`uint32_t`. 0 - if success 0xffffffff - if parameter invalid

NAME	writeTarget
DESCRIPTION	write data to target memory
SIGNATURE	`void writeTarget(uint32_t reg_id, uint32_t val)`
PARAMETERS	`reg_id` - id of the register that to be modified. `val` - value that should be written to register.
RETURN VALUE	`uint32_t`. 0 - if success 0xffffffff - if parameter invalid

NAME	hookFunctionPointer
DESCRIPTION	hook some function pointer for debug. This is for internal debug.
SIGNATURE	`void hookFunctionPointer(uint32_t func_id)`
PARAMETERS	`func_id` - id of the target function that to be hooked.
RETURN VALUE	`uint32_t`. 0 - if no target function others - real function pointer value

NAME	hookFunctionPointer
DESCRIPTION	hook some function pointer for debug. This is for internal debug.
SIGNATURE	`void hookFunctionPointer(uint32_t func_id)`
PARAMETERS	`func_id` - id of the target function that to be hooked.
RETURN VALUE	`uint32_t`. 0 - if no target function others - real function pointer value

References

[1] Thomas Anderson and Mike Dahlin. Operating systems principles & practice volume ii: Concurrency.

[2] Michael Kerrisk. *The Linux programming interface: a Linux and UNIX system programming handbook*. No Starch Press, 2010.