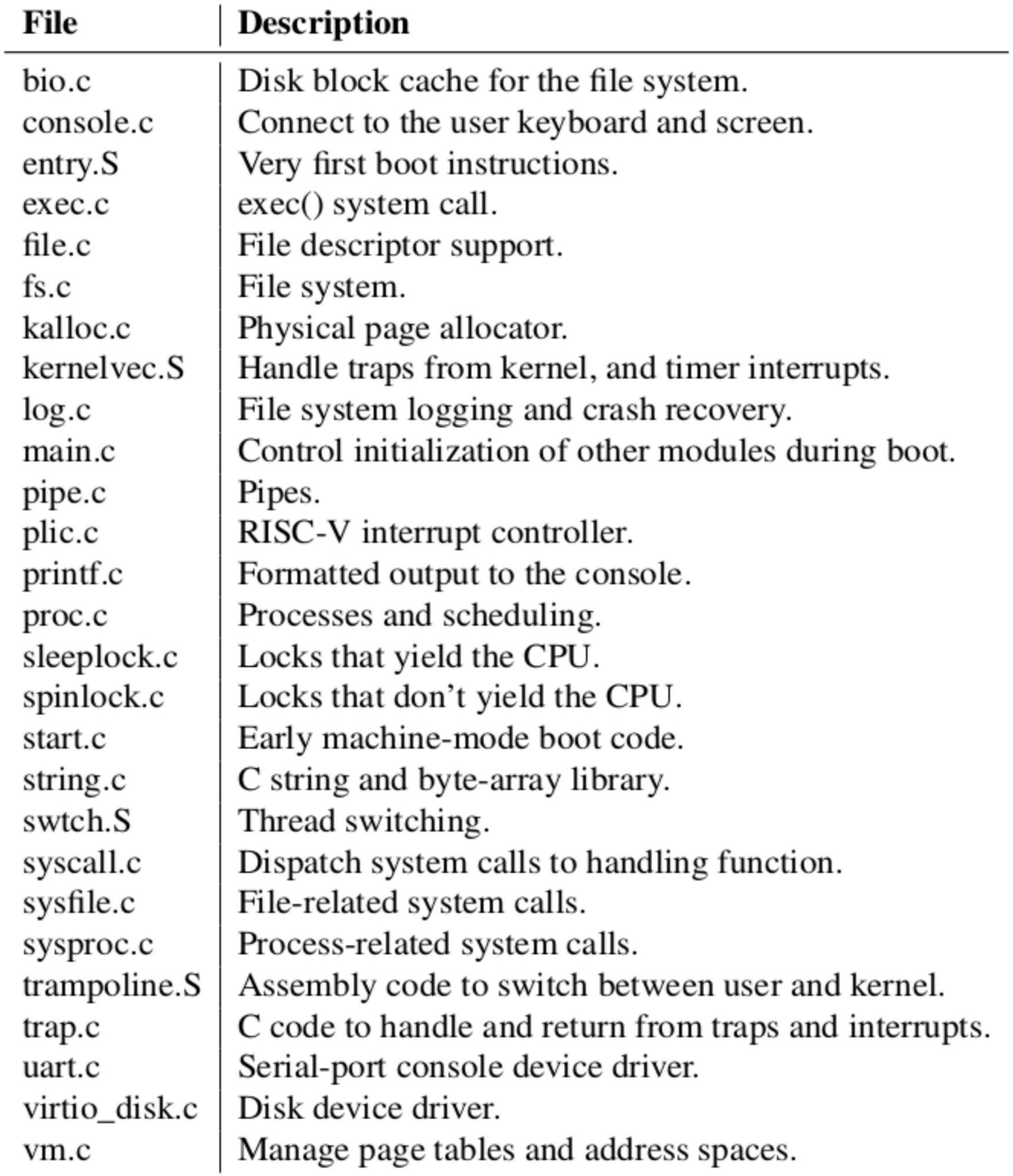
Xv6: introduzione

- Xv6[1] è stato sviluppato da Russ Cox, Frans Kaashoek e Robert Morris e viene usato per i corsi di SO del MIT
- Per l’installazione vedere: https://pdos.csail.mit.edu/6.828/2020/tools.html
- Xv6 gira su processori RISC-V multicore
- xv6 è scritto in “LP64” C:
  - long (L) e pointers (P) in linguaggio C sono a 64 bit
  - int a 32 bit
- Xv6 è sviluppato per supporto hardware simulato da qemu
  - Qemu è un SW open source per l’emulazione/virtualizzazione di architetture HW

[1] https://pdos.csail.mit.edu/6.828/2020/xv6.html

Xv6: organizzazione

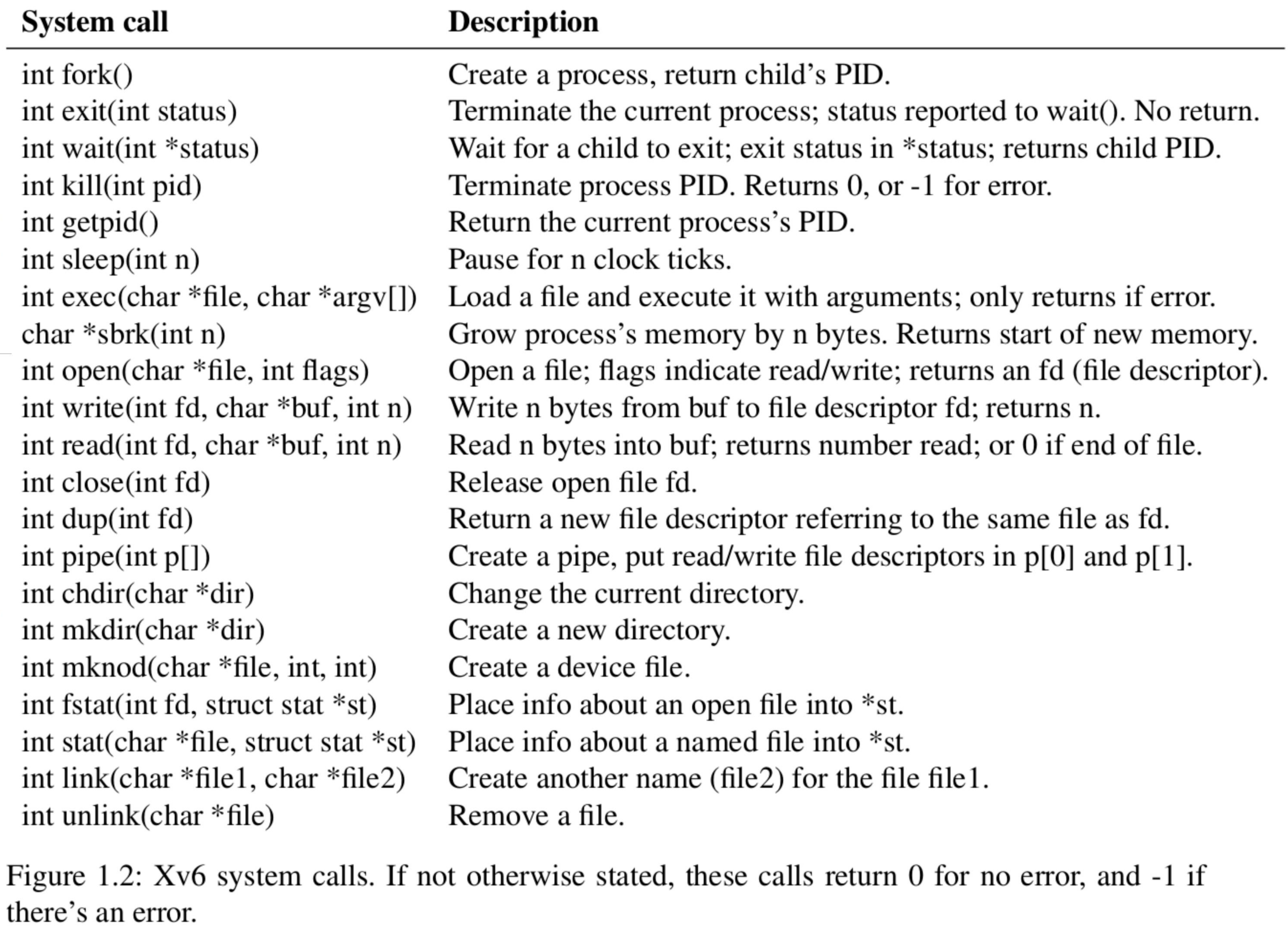
- Xv6 è sviluppato con un architettura monolitica
- E' costituito da 27 file sorgenti + 22 header per un totale di circa 10000 linee codice



File	Description
bio.c	Disk block cache for the file system.
console.c	Connect to the user keyboard and screen.
entry.S	Very first boot instructions.
exec.c	exec() system call.
file.c	File descriptor support.
fs.c	File system.
kalloc.c	Physical page allocator.
kernelvec.S	Handle traps from kernel, and timer interrupts.
log.c	File system logging and crash recovery.
main.c	Control initialization of other modules during boot.
pipe.c	Pipes.
plic.c	RISC-V interrupt controller.
printf.c	Formatted output to the console.
proc.c	Processes and scheduling.
sleeplock.c	Locks that yield the CPU.
spinlock.c	Locks that don't yield the CPU.
start.c	Early machine-mode boot code.
string.c	C string and byte-array library.
swtch.S	Thread switching.
syscall.c	Dispatch system calls to handling function.
sysfile.c	File-related system calls.
sysproc.c	Process-related system calls.
trampoline.S	Assembly code to switch between user and kernel.
trap.c	C code to handle and return from traps and interrupts.
uart.c	Serial-port console device driver.
virtio_disk.c	Disk device driver.
vm.c	Manage page tables and address spaces.

Figure 2.2: Xv6 kernel source files.

Xv6: system call

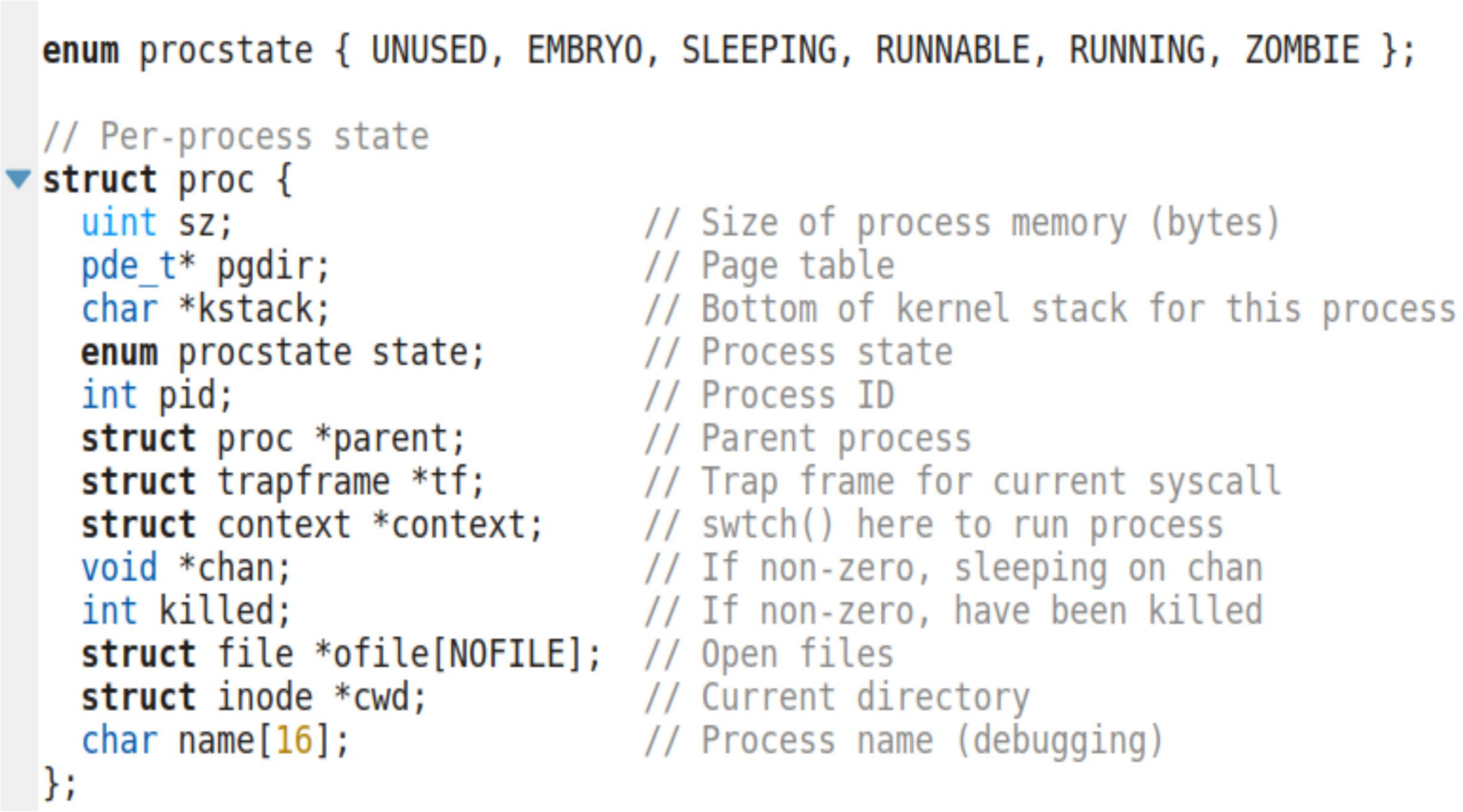


	System call	Description
sysproc.c	int fork()	Create a process, return child's PID.
	int exit(int status)	Terminate the current process; status reported to wait(). No return.
	int wait(int *status)	Wait for a child to exit; exit status in *status; returns child PID.
	int kill(int pid)	Terminate process PID. Returns 0, or -1 for error.
	int getpid()	Return the current process's PID.
	int sleep(int n)	Pause for n clock ticks.
	int exec(char *file, char *argv[])	Load a file and execute it with arguments; only returns if error.
	char *sbrk(int n)	Grow process's memory by n bytes. Returns start of new memory.
sysfile.c	int open(char *file, int flags)	Open a file; flags indicate read/write; returns an fd (file descriptor).
	int write(int fd, char *buf, int n)	Write n bytes from buf to file descriptor fd; returns n.
	int read(int fd, char *buf, int n)	Read n bytes into buf; returns number read; or 0 if end of file.
	int close(int fd)	Release open file fd.
	int dup(int fd)	Return a new file descriptor referring to the same file as fd.
	int pipe(int p[])	Create a pipe, put read/write file descriptors in p[0] and p[1].
	int chdir(char *dir)	Change the current directory.
	int mkdir(char *dir)	Create a new directory.
	int mknod(char *file, int, int)	Create a device file.
	int fstat(int fd, struct stat *st)	Place info about an open file into *st.
	int stat(char *file, struct stat *st)	Place info about a named file into *st.
	int link(char *file1, char *file2)	Create another name (file2) for the file file1.
	int unlink(char *file)	Remove a file.

Figure 1.2: Xv6 system calls. If not otherwise stated, these calls return 0 for no error, and -1 if there's an error.

Xv6: PCB entry

- Xv6 mantiene lo stato dei processi nella struttura proc (definita in proc.h) che comprende
  - Puntatore alla page table
  - Stato corrente, PID, genitore, file aperti del processo
  - Contesto per il context switch
- Ogni processo ha due stack utente e kernel separati e protetti
  - Quando un processo entra in modalità kernel il SO userà lo stack kernel del processo



```
enum procstate { UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };

// Per-process state
struct proc {
  uint sz;                     // Size of process memory (bytes)
  pde_t* pgdir;                // Page table
  char *kstack;                // Bottom of kernel stack for this process
  enum procstate state;        // Process state
  int pid;                     // Process ID
  struct proc *parent;         // Parent process
  struct trapframe *tf;        // Trap frame for current syscall
  struct context *context;     // swtch() here to run process
  void *chan;                  // If non-zero, sleeping on chan
  int killed;                  // If non-zero, have been killed
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;           // Current directory
  char name[16];               // Process name (debugging)
};
```

Xv6: layout memoria

- Xv6 usa HW page table per fornire a ciascun processo i suo spazio indirizzi virtuale
- La RISC-V page table mappa gli indirizzi virtuali (usati nelle istruzioni RISC-V) in indirizzi fisici
- Xv6 usa solo i 38 bit meno significativi per gli indirizzi virtuali, quindi
  - MAXVA=2^{38}-1= 0x3fffffff
  - definito in riscv.h

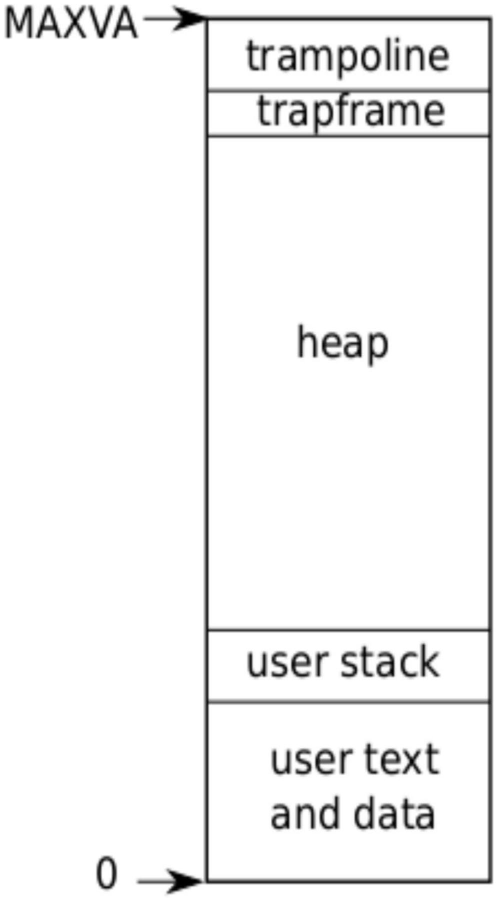


Figure 2.3: Layout of a process's virtual address space

Xv6: layout processo

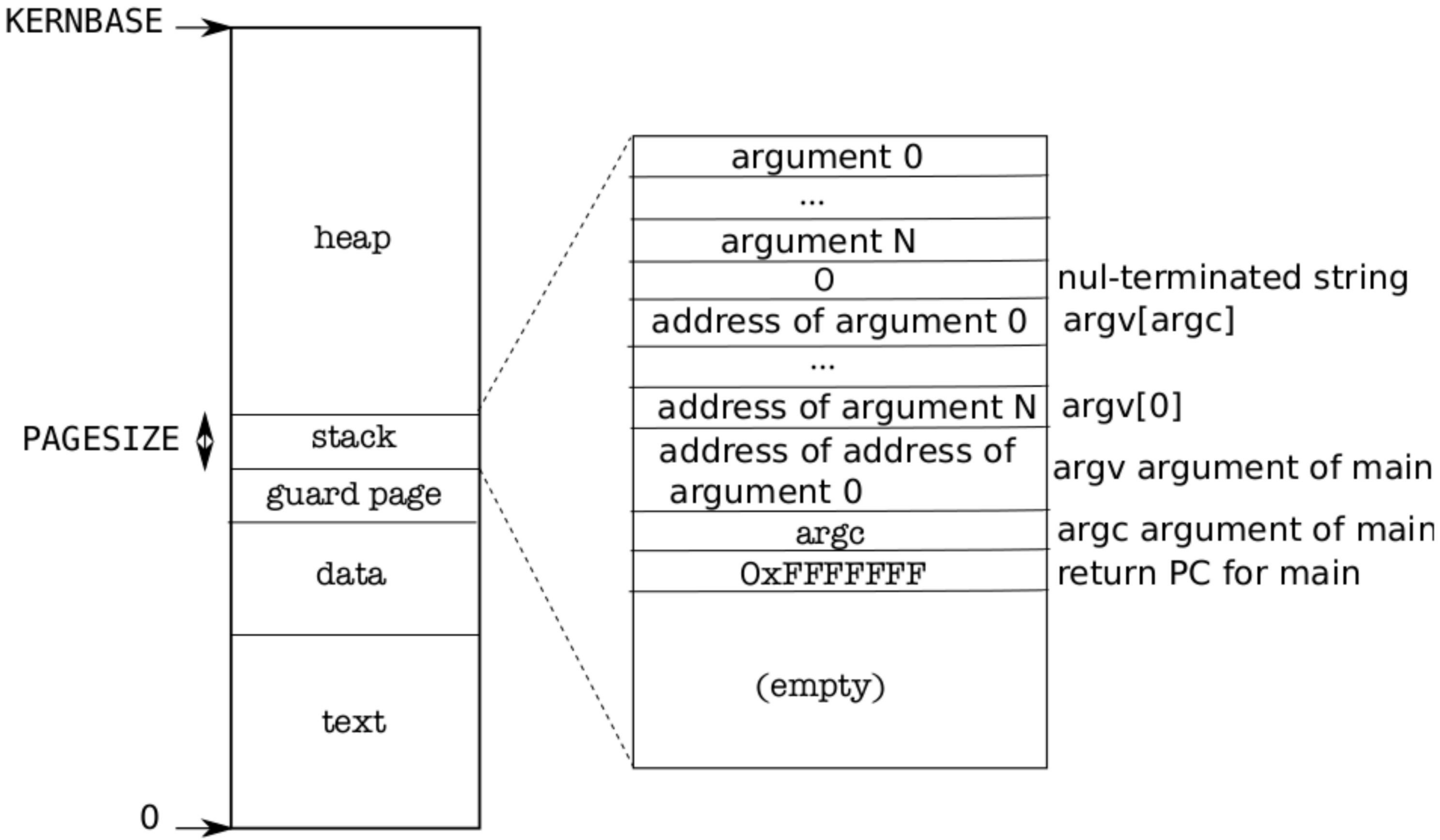


Figure 2-3. Memory layout of a user process with its initial stack.