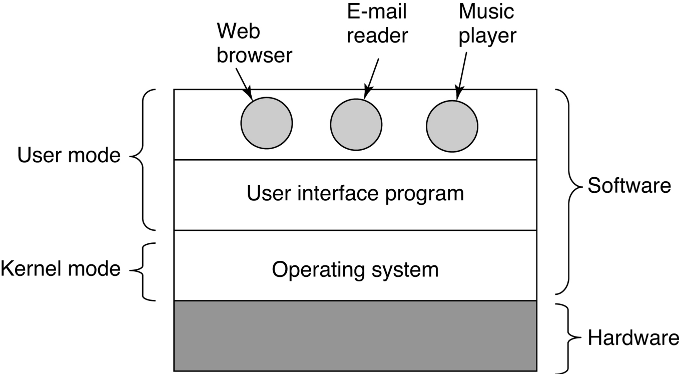
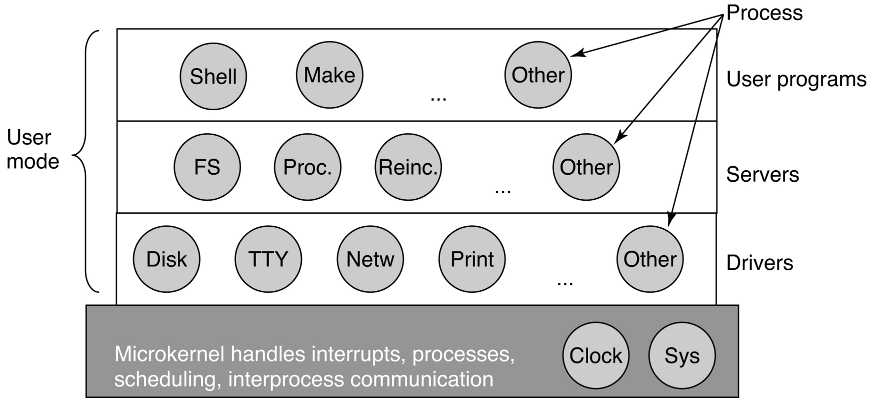
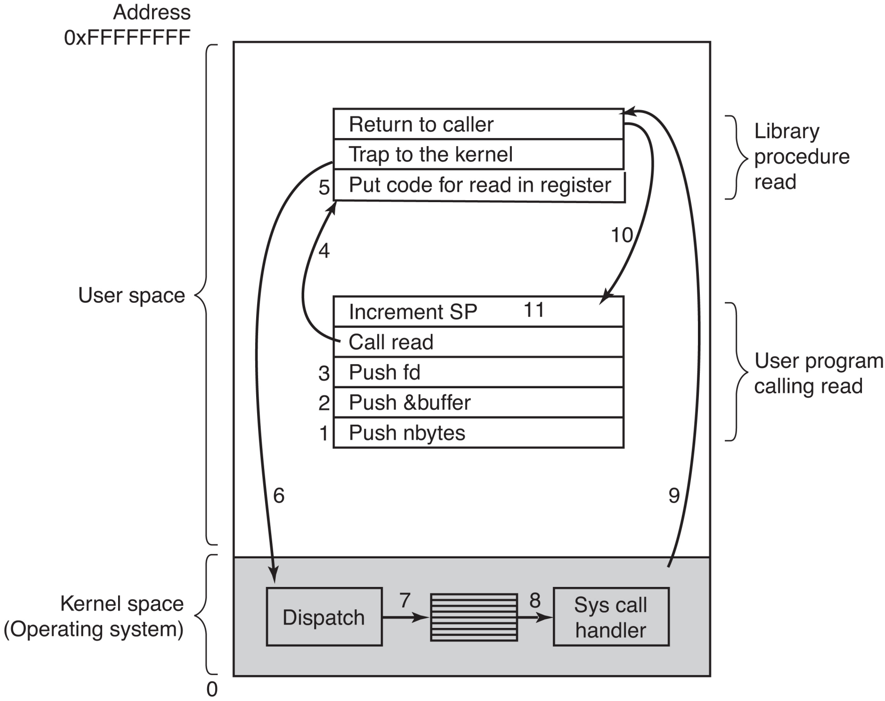

Web browser
E-mail reader
Music player
User mode
User interface program
Software
Kernel mode
Operating system
Hardware


Process
Shell
Make
...
Other
User programs
User mode
FS
Proc.
Reinc.
...
Other
Servers
Disk
TTY
Netw
Print
...
Other
Drivers
Microkernel handles interrupts, processes, scheduling, interprocess communication
Clock
Sys


Address
0xFFFFFFFF
User space
Kernel space
(Operating system)
0
Return to caller
Trap to the kernel
5 Put code for read in register
Library procedure read
10
4
Increment SP 11
Call read
3 Push fd
2 Push &buffer
1 Push nbytes
User program calling read
6
9
Dispatch
7
8
Sys call handler

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing, or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

Directory- and file-system management

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

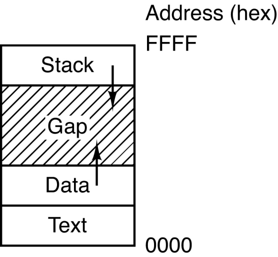
Miscellaneous

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970

```
/* Memory allocation example 1  */

#include <stdlib.h>
#include <stdio.h>

/* Allocate memory with the specified size (in bytes),
   returns zero upon failure */
void allocate (char** array, int size)
{
  *array = malloc (size);
}

void main (int argc, char* argv[])
{
  char* array;

  allocate (&array, 1024);
  if (!array)
  {
    fprintf(stderr, "Failed to allocate memory\n");
    return;
  }
  free (array);
}
```

```
/* Memory allocation example 2 */
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <errno.h>
#include <string.h>

void allocate (char** array, int size)
{
  *array = malloc (size);
}

void main (int argc, char* argv[])
{
  char* array;

  allocate (&array, (int) pow(2,30));
  if (!array)
  {
    fprintf(stderr, "*** %s\n", strerror(errno));
    return;
  }
  free (array);
}
```

```
#define TRUE 1

while (TRUE) {                                  /* repeat forever */
     type_prompt( );                            /* display prompt on the screen */
     read_command(command, parameters);         /* read input from terminal */

     if (fork( ) != 0) {                        /* fork off child process */
         /* Parent code. */
         waitpid(−1, &status, 0);               /* wait for child to exit */
     } else {
         /* Child code. */
         execve(command, parameters, 0);        /* execute command */
     }
}
```


Address (hex)
FFFF
Stack
Gap
Data
Text
0000

Exp.	Explicit	Prefix	Exp.	Explicit	Prefix
10^{-3}	0.001	milli	10^{3}	1,000	Kilo
10^{-6}	0.000001	micro	10^{6}	1,000,000	Mega
10^{-9}	0.000000001	nano	10^{9}	1,000,000,000	Giga
10^{-12}	0.000000000001	pico	10^{12}	1,000,000,000,000	Tera
10^{-15}	0.000000000000001	femto	10^{15}	1,000,000,000,000,000	Peta
10^{-18}	0.000000000000000001	atto	10^{18}	1,000,000,000,000,000,000	Exa
10^{-21}	0.000000000000000000001	zepto	10^{21}	1,000,000,000,000,000,000,000	Zetta
10^{-24}	0.000000000000000000000001	yocto	10^{24}	1,000,000,000,000,000,000,000,000	Yotta