User-Kernel Interface Documentation

1 Introduction

The user-kernel interface is a critical component of operating systems, allowing user-space applications to interact with kernel services. This interface is typically facilitated through system calls, which are requests from a program to the operating system to perform a specific task. The code files provided demonstrate various aspects of this interface, including process management, network communication, and signal handling.

2 Background

The code utilizes several libraries and frameworks, including:

- **stdio.h** for input/output operations
- **stdlib.h** for general-purpose functions
- **unistd.h** for system calls
- **sys/socket.h** for network socket operations
- **signal.h** for signal handling
- **ucontext.h** for handling segmentation faults

3 C Libraries Used

3.1 stdio.h

3.1.1 printf()

- **Function Signature:** `int printf(const char *format, ...)`
- **Arguments:** Format string followed by arguments to be formatted.
- **Return Value:** Number of characters printed.

3.1.2 scanf()

- **Function Signature:** `int scanf(const char *format, ...)`
- **Arguments:** Format string followed by pointers to store input values.
- **Return Value:** Number of successful assignments.

3.2 stdlib.h

3.2.1 exit()

- **Function Signature:** `void exit(int status)`
- **Arguments:** Exit status.
- **Return Value:** None.

3.3 unistd.h

3.3.1 getpid()

- **Function Signature:** `pid_t getpid(void)`
- **Arguments:** None.
- **Return Value:** Process ID.

3.3.2 syscall()

- **Function Signature:** `long syscall(long number, ...)`
- **Arguments:** System call number followed by arguments specific to the call.
- **Return Value:** Return value of the system call.

3.4 sys/socket.h

3.4.1 socket()

- **Function Signature:** `int socket(int domain, int type, int protocol)`
- **Arguments:** Domain, socket type, and protocol.
- **Return Value:** Socket file descriptor.

3.4.2 connect()

- **Function Signature:** `int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen)`
- **Arguments:** Socket file descriptor, address structure, and address length.
- **Return Value:** 0 on success, -1 on failure.

3.5 signal.h

3.5.1 sigaction()

- **Function Signature:** `int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact)`
- **Arguments:** Signal number, new action, and pointer to old action.
- **Return Value:** 0 on success, -1 on failure.

3.6 ucontext.h

3.6.1 siginfo_t

- **Structure:** Contains information about the signal that caused the context switch.
- **Members:** Various fields depending on the signal type.

4 Programs

4.1 innocent.c

This program prints the process ID of the running process and enters an infinite loop. It demonstrates basic process management by using the `getpid()` function to retrieve the process ID.

4.1.1 Conclusion

This file is a simple demonstration of process ID retrieval and does not perform any complex operations.

4.2 netlink.c

This program uses Netlink sockets to listen for process events such as fork, exec, and exit. It sets up a Netlink socket, binds it to a specific group, and sends a subscription message to receive these events.

4.2.1 Conclusion

This file showcases how to use Netlink sockets for monitoring process events, providing insights into system activity.

4.3 syscall.c

This program prompts the user to enter a system call number and then executes that system call using the `syscall()` function. It demonstrates how to invoke system calls directly from user space.

4.3.1 Conclusion

This file illustrates the direct invocation of system calls, allowing users to interact with kernel services at a low level.

4.4 segfault.c

This program intentionally causes a segmentation fault by dereferencing a NULL pointer and sets up a signal handler to catch the resulting SIGSEGV signal. It prints the register state and backtrace at the time of the fault.

4.4.1 Conclusion

This file demonstrates how to handle segmentation faults and provides diagnostic information about the fault, aiding in debugging.

4.5 socket.c

This program establishes a TCP connection to a server (in this case, baidu.com) and sends an HTTP GET request. It receives and prints the server's response.

4.5.1 Conclusion

This file shows how to create a network connection and communicate with a server using sockets, illustrating basic network programming.

4.6 interruptme.c

This program sets up a signal handler for SIGINT (interrupt signal) and counts how many times it is received. After two SIGINT signals, the program exits.

4.6.1 Conclusion

This file demonstrates signal handling and how to manage program termination based on external signals.