Linux Programming
Python System Programming

컴퓨터공학과 송석일
sisong@ut.ac.kr

Python LInux System Programming

- 파일 및 디렉터리 관리
  - os 모듈, shutil 모듈 : 파일과 디렉터리를 생성, 삭제, 이동, 복사
- 프로세스 관리
  - subprocess 모듈 : 새로운 프로세스 생성, 프로세스 입출력을 제어
  - os 모듈 : fork, exec 함수 사용 프로세스 제어
- 네트워킹
  - socket 모듈 : TCP/IP 및 UDP 소켓 프로그래밍 수행
- 시그널 처리
  - signal 모듈 : 시그널을 처리하고, 특정 시그널에 대한 핸들러 정의
- 인터프로세스 통신 (IPC)
  - multiprocessing 모듈 : 파이프, 큐 등 프로세스 간 통신
- 메모리 매핑 파일
  - mmap 모듈 : 메모리 매핑 파일
- 시스템 호출:
  - os 모듈을 사용하여 리눅스 시스템 호출을 직접 호출
  - 예) os.open, os.read, os.write 등

파일 관련

```
import os

# 파일 열기
fd = os.open('example1.txt', os.O_RDWR | os.O_CREAT)

# 파일에 쓰기
os.write(fd, b'Hello, this is a test.')

# 파일 포인터를 처음으로 이동
os.lseek(fd, 0, os.SEEK_SET)

# 파일에서 읽기
data = os.read(fd, 100)
print(f'Read data: {data.decode()}')

# 파일 포인터를 특정 위치로 이동
os.lseek(fd, 7, os.SEEK_SET)

# 파일에서 읽기
data = os.read(fd, 100)
print(f'Read data from position 7: {data.decode()}')

# 파일 닫기
os.close(fd)
```

```
import os

# 파일 열기 또는 생성
file_path = 'example2.txt'
flags = os.O_RDWR | os.O_CREAT  # 읽기/쓰기 모드 및 파일이 없으면 생성
mode = 0o666  # 파일 권한 (rw-rw-rw-)

# 파일 디스크립터 생성
fd = os.open(file_path, flags, mode)

# 파일에 쓰기
os.write(fd, b'Hello, this is a test created using os.open().')

# 파일 포인터를 처음으로 이동
os.lseek(fd, 0, os.SEEK_SET)

# 파일에서 읽기
data = os.read(fd, 100)
print(f'Read data: {data.decode()}')

# 파일 닫기
os.close(fd)
```

Directory 관련

```
import os

# 디렉토리 생성
dir_name = 'example_dir'
if not os.path.exists(dir_name):
    os.mkdir(dir_name)
    print(f"Directory '{dir_name}' created.")
else:
    print(f"Directory '{dir_name}' already exists.")

# 디렉토리 변경
os.chdir(dir_name)
print(f"Changed current directory to '{dir_name}'.")

# 파일 생성
fd=os.open('example1.txt', os.O_RDWR | os.O_CREAT,
0o666 )
os.close(fd)

# 부모 디렉토리로 이동하여 디렉토리 내용 읽기
os.chdir('..’)
# 디렉토리 열기 및 내용 읽기
with os.scandir(dir_name) as entries:
    print(f"Contents of directory '{dir_name}':")
    for entry in entries:
        print(entry.name)

# 디렉토리 제거
if os.path.exists(dir_name):
    os.rmdir(dir_name)
    print(f"Directory '{dir_name}' removed.")
else:
    print(f"Directory '{dir_name}' does not exist.")
```

Process 관련

```python
import os

def child_process():
    print("Child process (PID: {})".format(os.getpid()))
    # 새로운 프로그램 실행
    os.execlp("ls", "ls", "-l")

def parent_process(child_pid):
    print("Parent process (PID: {})".format(os.getpid()))
    print("Waiting for child process (PID: {}) to finish...".format(child_pid))
    os.waitpid(child_pid, 0)
    print("Child process finished.")

def main():
    print("Main process (PID: {})".format(os.getpid()))
    pid = os.fork()

    if pid == 0:
        # 자식 프로세스
        child_process()
    else:
        # 부모 프로세스
        parent_process(pid)

if __name__ == "__main__":
    main()
```

Process 관련

```python
import os
import time

def child_process():
    print("Child process (PID: {}) started.".format(os.getpid()))
    time.sleep(5)  # 자식 프로세스가 5초 동안 실행됨
    print("Child process (PID: {}) finished.".format(os.getpid()))
    os._exit(0)
def parent_process(child_pid):
    print("Parent process (PID: {}) waiting for child process.".format(os.getpid()))

    # 자식 프로세스가 종료될 때까지 블록 (0 옵션)
    pid, status = os.waitpid(child_pid, 0)
    print(f"Child process {pid} terminated with status {status}.")
def non_blocking_wait(child_pid):
    print("Non-blocking wait in parent process (PID: {}).".format(os.getpid()))

    # 자식 프로세스가 종료되지 않더라도 즉시 반환 (os.WNOHANG 옵션)
    pid, status = os.waitpid(child_pid, os.WNOHANG)
    if pid == 0:
        print("Child process has not terminated yet.")
    else:
        print(f"Child process {pid} terminated with status {status}.")
def main():
    print("Main process (PID: {})".format(os.getpid()))
    pid = os.fork()

    if pid == 0:
        # 자식 프로세스
        child_process()
    else:
        # 부모 프로세스
        non_blocking_wait(pid)  # 비블로킹 대기
        print("Parent can do something while wait for child")
        parent_process(pid)  # 블로킹 대기
if __name__ == "__main__":
    main()
```

Process 관련

```python
import os
import time

def child_process(index):
    print("Child process {} (PID: {}) started.".format(index, os.getpid()))
    time.sleep(5 + index)  # 자식 프로세스가 실행되는 시간 (5초 + index)
    print("Child process {} (PID: {}) finished.".format(index, os.getpid()))
    os._exit(0)
def parent_process(child_pid):
    print("Parent process (PID: {}) waiting for child process {}.".format(os.getpid(), child_pid))
    # 자식 프로세스가 종료될 때까지 블록 (0 옵션)
    pid, status = os.waitpid(child_pid, 0)
    print(f"Child process {pid} terminated with status {status}.")
def main():
    print("Main process (PID: {})".format(os.getpid()))
    # 첫 번째 자식 프로세스 생성
    pid1 = os.fork()
    if pid1 == 0:
        # 첫 번째 자식 프로세스
        child_process(1)
    # 두 번째 자식 프로세스 생성
    pid2 = os.fork()
    if pid2 == 0:
        # 두 번째 자식 프로세스
        child_process(2)

    # 부모 프로세스는 두 자식 프로세스의 종료를 차례로 기다림
    parent_process(pid1)
    parent_process(pid2)

if __name__ == "__main__":
    main()
```

dup2

```python
import os
import sys

# 리디렉션할 파일 열기
filename = 'output.txt'
with open(filename, 'w') as f:
    # 현재 stdout의 파일 디스크립터 저장
    saved_stdout = os.dup(1)

    # stdout을 파일 디스크립터로 복제하여 리디렉션
    os.dup2(f.fileno(), 1)

    try:
        # 이제 stdout이 파일로 리디렉션됨
        print("This will be written to the file.")
    finally:
        # 원래 stdout으로 복구
        os.dup2(saved_stdout, 1)
        os.close(saved_stdout)

print("This will be printed on the console.")
```

mmap

```python
import mmap
import os

# 샘플 파일 생성
filename = 'sample.txt'
with open(filename, 'wb') as f:
    f.write(b'Hello, this is a sample text.')

# 파일을 메모리에 매핑하여 읽기 및 쓰기
with open(filename, 'r+b') as f:
    # 파일 크기 가져오기
    filesize = os.path.getsize(filename)

    # 파일을 메모리에 매핑
    with mmap.mmap(f.fileno(), length=filesize, access=mmap.ACCESS_WRITE) as m:
        # 매핑된 메모리에서 데이터 읽기
        original_content = m[:]
        print(f'Original content: {original_content.decode("utf-8")}')

        # 매핑된 메모리의 데이터 수정
        m[0:2] = b'Hi'
        m[2:] = b', this is the updated text.'

        # 수정된 내용 확인
        updated_content = m[:]
        print(f'Updated content: {updated_content.decode("utf-8")}')

# 수정된 파일 내용 확인
with open(filename, 'rb') as f:
    print(f'Final file content: {f.read().decode("utf-8")}')
```

Signal 관련

```python
import signal
import time

def signal_handler(signum, frame):
    print(f"Signal {signum} received. Exiting gracefully...")
    exit(0)

# SIGINT 신호에 대한 핸들러 설정
signal.signal(signal.SIGINT, signal_handler)

print("Press Ctrl+C to trigger the signal handler.")
while True:
    print("Running...")
    time.sleep(1)
```

pipe

```python
import os

def child_process(pipe_out):
    os.close(pipe_out[0])  # 읽기 끝 닫기
    w = os.fdopen(pipe_out[1], 'w')  # 쓰기 끝 파일 객체로 열기
    w.write("Hello from child process!")
    w.close()

def parent_process(pipe_in):
    os.close(pipe_in[1])  # 쓰기 끝 닫기
    r = os.fdopen(pipe_in[0])  # 읽기 끝 파일 객체로 열기
    message = r.read()
    print(f"Parent process received: {message}")
    r.close()

def main():
    pipe_in, pipe_out = os.pipe()  # 파이프 생성

    pid = os.fork()  # 새로운 프로세스 생성

    if pid == 0:
        # 자식 프로세스
        child_process((pipe_in, pipe_out))
    else:
        # 부모 프로세스
        parent_process((pipe_in, pipe_out))

if __name__ == "__main__":
    main()
```

```python
import os

def child_process(pipe_out):
    os.close(pipe_out[0])  # 읽기 끝 닫기
    os.write(pipe_out[1], b"Hello from child process!")
    os.close(pipe_out[1])  # 쓰기 끝 닫기

def parent_process(pipe_in):
    os.close(pipe_in[1])  # 쓰기 끝 닫기
    message = os.read(pipe_in[0], 1024)
    print(f"Parent process received: {message.decode()}")
    os.close(pipe_in[0])  # 읽기 끝 닫기

def main():
    pipe_in, pipe_out = os.pipe()  # 파이프 생성

    pid = os.fork()  # 새로운 프로세스 생성

    if pid == 0:
        # 자식 프로세스
        child_process((pipe_in, pipe_out))
    else:
        # 부모 프로세스
        parent_process((pipe_in, pipe_out))

if __name__ == "__main__":
    main()
```

Named pipe

reader.py

```python
import os

fifo = 'mypipe'

# Named pipe에서 데이터를 읽기
with open(fifo, 'r') as pipe:
    while True:
        message = pipe.readline()
        if message:
            print(f"Read: {message.strip()}")
        else:
            break
```

writer.py

```python
import os
import time

fifo = 'mypipe'

# Named pipe에 데이터를 쓰기
with open(fifo, 'w') as pipe:
    for i in range(5):
        message = f"Message {i}"
        print(f"Writing: {message}")
        pipe.write(message + '\n')
        pipe.flush()
        time.sleep(1)
```

위의 코드를 통해 입력과 쓰기를 하려면 네임드 파이프를 먼저 생성해야 함

$ mknod mypipe p

문제 1

두개의 자식 프로세스를 생성하고 두 자식 프로세스가 파이프를 통해 메시지를 송수신하는 프로그램을 작성하라.
첫번째 생성된 자식 프로세스는 “Hello” 를 전송하고 두번째 생성된 자식 프로세스는 “Hello”를 수신하고 출력하도록 하라.
부모 프로세스는 두 자식 프로세스가 모두 종료될때 까지 대기했다가 자식 프로세스들이 종료되면 종료되었음을 출력하라.

문제 2

두개의 자식 프로세스를 생성하고 두 자식 프로세스가 파이프를 통해 메시지를 송수신하는 프로그램을 작성하라.
첫번째 생성된 자식 프로세스는 ls 명령을 실행하고 이때 생성된 결과를 두번째 자식 프로세스로 파이프를 통해서 전송한다.
두번째 생성된 자식 프로세스는 wc명령을 실행하고 wc 명령의 입력은 첫번째 자식 프로세스가 전송하는 ls 명령의 실행결과를 파이프를 통해서 입력받아 실행되도록 한다.