1 Assignment7 Report

1.1 Some Background

1. PID (Process ID): Unique identifier for each process.

2. PGRP (Process Group ID): ID of the process group. A process group is a collection of one or more processes.

3. TPGID (Terminal Process Group ID): ID of the foreground process group of the controlling terminal. If a process does not have a controlling terminal, this will be 0 (in Unix environment, like BSD, Mac), or -1 (in Linux environemnet, tested on Ubuntu).

1.2 Implementation

When we *fork()* a process, the return value of *fork()* is the PID of the child process in the parent process, and 0 in the child process. So we can use this to distinguish the parent process and the child process.

In the child process, we can use *setsid()* to create a new session.

Because the child process has its own session, so it does not have a controlling terminal, and it'll be the leader of the process group.

The value of PID, PGID will be the same in the child process, and the value of TPGID will be 0 (in Unix environment, like BSD, Mac), or -1 (in Linux environemnet, tested on Ubuntu).

1.3 Codes

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main() {
    pid_t pid = fork();
    if (pid == 0) {
        // child process
        pid_t sid = setsid();
        if (sid < 0) {
            printf("Error: cannot create a new session\n");
            return -1;
        }

        printf("child process pid: %d, pgid: %d, sid: %d\n", getpid(),
            getpgid(getpid()), sid);

        FILE *fp;
        char str[1024];
        char real_cmd[1024];
        char *cmd = "ps -x -o pid,pgid,tpgid";
        sprintf(real_cmd, cmd, getpgid(getpid()));

        fp = popen(real_cmd, "r");
        if (fp == NULL) {
            printf("popen failed\n");
            return -1;
        }
        while (fgets(str, sizeof(str), fp) != NULL) {
            printf("%s", str);
        }
        pclose(fp);
    } else if (pid > 0) {
        // parent process
        // semaphore wait
        printf("parent process pid: %d, pgid: %d, sid: %d\n", getpid(),
            getpgid(getpid()), getsid(getpid()));
        wait(NULL);
    } else {
        perror("Error: cannot fork a new process\n");
        return -1;
    }
    return 0;
}
```

程式碼 **1.1:** assignment7.c

```
# One FreeBSD 13.2-RELEASE-p4 FreeBSD 13.2-RELEASE-p4 GENERIC amd64
ryanchang1117@freebsd-13-1:~/APUE_assignment7 $ ./assignment7
parent process pid: 1256, pgid: 1256, sid: 903
child process pid: 1257, pgid: 1257, sid: 1257
 PID PGID TPGID
 902  900     0
1257 1257     0
1258 1257     0
 903  903  1256
1256 1256  1256
```

程式碼 **1.2:** 指令紀錄 (On BSD)

```
# On Linux 5.4.0-166-generic #183-Ubuntu SMP Mon Oct 2 11:28:33 UTC 2023
    x86_64 GNU/Linux
hsuan@t1:~/APUE_assignment7$ ./assignment7
parent process pid: 767550, pgid: 767550, sid: 767446
child process pid: 767551, pgid: 767551, sid: 767551
    PID    PGID   TPGID
 767365  767365      -1
 767368  767365      -1
 767373  767373      -1
 767392  767392      -1
 767445  767344      -1
 767446  767446  767550
 767489  767489  767550
 767550  767550  767550
 767551  767551      -1
 767552  767551      -1
 767553  767551      -1
```

程式碼 **1.3:** 指令紀錄 (On Ubuntu)

```
CC = gcc
CFLAG = -std=c11 -O2 -Wall
TARGET = assignment7
SRCS = assignment7.c
OBJS = assignment7.o

RPT_FILES := report.tex
PDF_FILES := report.pdf

all: clean $(TARGET)

$(TARGET): $(OBJS)
  $(CC) $(CFLAG) -o $(TARGET) $(OBJS)

%.o: %.c
  $(CC) $(CFLAGS) -c $< -o $@

- pdf:
  docker run -v $(shell pwd):/code -it --rm --name xelatex-build lfswang/
      xelatex:latest sh /code/run.sh

clean:
  rm -f $(OBJS) $(TARGET) *.toc *.synctex.gz *.out *.log *.aux *.lot *.
      lof *.bcf *.run.xml *.pdf
```

程式碼 **1.4:** Makefile