Hà Tấn Linh
MSSV: 1512284

HỆ ĐIỀU HÀNH NÂNG CAO
Đồ án 2: Cài đặt Schiper-Eggli-Sandoz Algorithm

BÁO CÁO ĐỒ ÁN

I. Thông tin cơ bản

- Môi trường: ***GNU/Linux***.
- Ngôn ngữ lập trình: ***C (POSIX Threads)***.
- Công cụ: ***gcc***, ***make***, ***bash***.
- Cấu trúc thư mục - bên trong thư mục đồ án gồm:
  + `src`: chứa **mã nguồn** chương trình và tập tin kịch bản biên dịch **makefile**. Thư mục con `include` chứa các tập tin **header**.
  + `test`: chứa các tập tin **config** mẫu để chạy thử chương trình; cũng như các tập tin **shell scripts** dùng để tự động chạy nhiều process cùng lúc. Các thư mục con có tên dạng `logs_*` chứa các tập tin **log** khi chạy chương trình với các file config mẫu.
  + Tập tin `README.pdf`: báo cáo và hướng dẫn sử dụng đồ án.

II. Biên dịch và thử nghiệm chương trình

1. *Biên dịch:*

- Để biên dịch chương trình, ở trong thư mục `src` chạy lệnh `make`. Khi đó thư mục `obj` sẽ được tự động tạo để chứa các object files được biên dịch ra. Cuối cùng các object files sẽ được link lại thành tập tin chương trình **`1512284`** ở thư mục `src`.
- Có thể dùng lệnh `make debug=true` để bật chế độ biên dịch DEBUG.
- Ngoài ra, trong môi trường localhost, các gói tin được gửi thường sẽ đến ngay đích đến mà không có độ trễ, điều này dẫn đến tính năng buffer message khó được thể hiện ra. Khi đó có thể biên dịch với lệnh `make delay=true` để bật tính năng giả lập độ trễ trên đường truyền mạng.
- Hoặc, có thể kết hợp cả hai: `make debug=true delay=true`
- Để dọn dẹp các tập tin được biên dịch ra - dùng lệnh `make clean`.

2. *Config:*

- Khi chạy chương trình, tham số dòng lệnh thứ nhất sẽ là đường dẫn đến tập tin config cho process đó, nếu không có tham số dòng lệnh này, process sẽ nhận giá trị mặc định.

- Cấu trúc tập tin config: gồm nhiều dòng, mỗi dòng chỉ định giá trị cho một thuộc tính, thứ tự các dòng không quan trọng, một thuộc tính có thể được định nghĩa nhiều lần, lúc đó thuộc tính nhận giá trị cuối cùng được định nghĩa.
- Cú pháp chung: `property:value`
- Các thuộc tính có thể định nghĩa:
  + *`n`*: Số lượng process trong hệ thống. Mặc định `n = 15`.
  + *`id`*: Số thứ tự của process trong hệ thống. Mặc định `id = 0`.
  + *`nmsg`*: Số lượng tin nhắn gửi đi (cho mỗi process khác). Mặc định `nmsg = 150`.
  + *`rmin`*: Tần suất gửi tin tối thiểu (Đơn vị: message/phút). Mặc định: `rmin = 40`.
  + *`rmax`*: Tần suất gửi tin tối đa (Đơn vị: message/phút). Mặc định `rmax = 100`.
  + *`lport`*: Cổng lắng nghe của process. Mặc định `lport = 1400`.
  + *`log`*: Đường dẫn đến tập tin log của process. Mặc định: `log = process_{id}_log.txt`.
  + *`process:X:Y:Z`* - Thông tin truy cập của process `X` trong hệ thống. Mặc định: `Y = 127.0.0.1`, `Z = 1400 + X`.

3. *Chạy thử: (Giả định đang ở trong thư mục `test`)*
   - Chạy chương trình bình thường: `../src/1512284 <tên config file>`. Lúc này chương trình sẽ khởi tạo socket lắng nghe trên *`lport`* sau đó dừng lại và chờ. Để chương trình bắt đầu gửi tin nhắn đến các process khác, nhập lệnh `start` vào stdin của chương trình hoặc truyền signal `SIGUSR1` cho chương trình bằng lệnh `kill -s USR1 <process id>`.
   - Chạy chương trình bằng các shell scripts được viết sẵn:
     + 2 process: `./autorun_2p.sh ../src/1512284`
     + 4 process: `./autorun_4p.sh ../src/1512284`
     + 15 process: `./autorun_15p.sh ../src/1512284`

(**Lưu ý:** các shell scripts chỉ chạy được khi được gọi bên trong thư mục *`test`*)

III. Thiết kế và cài đặt

1. *Thiết kế:*
   - Khi bắt đầu, chương trình khởi tạo giá trị mặc định cho các thuộc tính; rồi đọc tập tin config để ghi đè lên giá trị mặc định, nếu tham số dòng lệnh được cung cấp.
   - Sau đó, chương trình khởi tạo vector clock và vector gói tin đã gửi với giá trị 0.
   - Tiếp theo, chương trình sẽ dừng ở trạng thái chờ cho đến khi người dùng nhập lệnh `start` hoặc nó nhận được signal `SIGUSR1` rồi mới bắt đầu chạy tiếp.
   - Lúc này, chương trình khởi tạo *`n-1`* threads để gửi tin nhắn đến các process khác một cách đồng thời. Nếu macro `_TEST_DELAY_MESSAGE_` được định nghĩa (biên dịch bằng `make delay=true`), thì với mỗi message muốn gửi,

thread hiện hành sẽ phải tạo một thread mới, thread mới này delay ngẫu nhiên từ 0 đến 3 giây rồi mới thực sự gửi message đi.

- Cuối cùng, chương trình dùng file descriptor polling (thủ tục `select()`) để xử lý tuần tự các messages nhận vào và quyết định delivery hay buffer message bằng thuật toán SES.
- Khi một thread gửi đủ *nmsg* cho một process khác, nó sẽ gửi thêm một message có nội dung `"FIN:SEND"` đến process đó. Chương trình chỉ kết thúc khi đã nhận và delivery đủ *n-1* message `"FIN:SEND"`.

2. *Cài đặt:*

- *main.c*: Entry point của chương trình, define các global variables và quản lý việc bắt đầu gửi message bằng signal hoặc user input.
- *defs.h*: Declare các global variables và các kiểu dữ liệu chính của chương trình.
- *timevec.c* và *timevec.h*: Kiểu dữ liệu biểu diễn vector clock và các hàm liên quan. Vector clock được cài đặt là một mảng các số nguyên được cấp phát động.
- *sentvec.c* và *sentvec.h*: Kiểu dữ liệu biểu diễn vector chứa thông tin các message đã gửi. Là một danh sách liên kết.
- *config.c* và *config.h*: Các thủ tục gán giá trị mặc định cho các thuộc tính cũng như đọc config file.
- *logs.c* và *logs.h*: Các thủ tục ghi log file và xuất dữ liệu ra màn hình.
- *threads.c* và *threads.h*: Cài đặt việc khởi tạo và thực thi các threads để gửi message đến process khác; cũng như cài đặt tính năng giả lập độ trễ đường truyền.
- *ses.c* và *ses.h*: Cài đặt việc nhận và quản lý thứ tự nhân quả của các message bằng thuật toán SES.