Message Dispatcher

Creare un file "`msgDispatcher.c`" e un makefile che genera il corrispondente eseguibile "`msgDispatcher.out`". Tale programma dovrà gestire la comunicazione con dei processi figli creati dinamicamente. In particolare, il programma dovrà accettare da *stdin* dei comandi, i quali potranno essere interpretati come stringhe (es: "ciao come stai") o interi positivi (es: "32"). Se il comando equivale ad un intero positivo allora il programma dovrà creare, a prescindere dal valore di tale numero, un (uno solo) nuovo figlio fino ad un massimo definito con una direttiva `#define` impostata a 5 come base di riferimento, mentre se il comando equivale a una stringa allora essa dovrà essere salvata in memoria (si mantiene solo l'ultima inserita).
Il programma dovrà poi supportare la ricezione di due segnali: SIGUSR1 e SIGUSR2. Alla ricezione di uno qualunque dei due segnali il programma dovrà delegare un thread per l'invio di un messaggio a tutti i figli attivi. Il messaggio, che dovrà essere inviato attraverso pipes anonime, dovrà contenere il PID del figlio a cui viene mandato, l'indice *i* (dove i = 0 se il figlio è stato il primo ad essere creato e così via...), e l'ultima stringa ricevuta dall'utente tramite comando.
Ogni figlio, una volta ricevuto il messaggio personalizzato, dovrà stamparlo e terminare (lasciando dunque al programma la possibilità di generare ulteriori figli: "reset" dell'indice *i*).
Ogni thread dovrà inoltre documentare in un file di log (/tmp/log.txt) il numero del segnale che ha portato alla sua generazione ed eventuali altre operazioni svolte.

Nel particolare il programma dovrà implementare le seguenti funzionalità

- Operazioni di routine:
  - Stampare il proprio PID come prima riga
  - Gestire la ricezione di comandi su stdin, interi positivi o stringhe.
  - Gestire la chiusura con CTRL+C per terminare i figli attivi.
- Gestione segnali:
  - Impostare la ricezione dei segnali SIGUSR1 e SIGUSR2
- Gestione Thread
  - Generare un nuovo thread ad ogni nuovo segnale (SIGUSR1/2)
  - Scrivere su un file di log le operazioni del thread
  - Invio ai figli attivi di un messaggio (con pipe anonime) contenente l'ultima stringa inviata dall'utente, l'indice del figlio ed il suo pid.
  - Terminare il thread a fine invio
- Gestione figli:
  - Generare un nuovo figlio ogni volta che in input si ha un intero positivo
  - Gestire la ricezione di messaggi su pipe anonime
  - Ogni figlio dovrà terminare dopo la ricezione e stampa del messaggio ricevuto
  - Consentire l'esistenza contemporanea di un certo numero massimo di figli

Esempio:

```
[MAIN] main PID is 231
7
[CHD] I'm a new child with PID 232, and I'm waiting for msg from my father
21
[CHD] I'm a new child with PID 233, and I'm waiting for msg from my father
first text
[MAIN] Msg 'first text' saved
[THREAD] msg='To Child 0 PID 232: first text'
[THREAD] msg='To Child 1 PID 233: first text'
[CHD] I received the following message: 'To Child 0 PID 232: first text'
[CHD] I received the following message: 'To Child 1 PID 233: first text'
new text
[MAIN] Msg 'new text' saved
9
[CHD] I'm a new child with PID 236, and I'm waiting for msg from my father
[THREAD] msg='To Child 0 PID 236: new text'
[CHD] I received the following message: 'To Child 0 PID 236: new text'
```

- Esternamente è inviato un segnale `SIGUSR1`: si crea un thread che non invia però alcun messaggio (non essendoci ancora figli).
- All'input "`21`" (intero positivo) si genera un secondo figlio.
- All'input "`first text`" si memorizza la stringa (sovrascrivendo la precedente).
- Esternamente è inviato un segnale `SIGUSR1`: si crea un thread che invia un messaggio con i dati richiesti ai due figli, i quali terminano.
- All'input "`new text`" si memorizza la stringa (sovrascrivendo la precedente).
- All'input "`9`" (intero positivo) si genera un nuovo figlio.
- Esternamente è inviato un segnale `SIGUSR2`: si crea un thread che invia un messaggio con i dati richiesti al figlio, il quale termina.
- Esternamente è inviato un segnale `SIGINT` e il programma termina.

Il file /tmp/log.txt contiene:

```
[THREAD] I'm a new thread generated by signal 10
[THREAD] Sending message to children
[THREAD] children reset
[THREAD] Terminating thread
[THREAD] I'm a new thread generated by signal 10
[THREAD] Sending message to children
[THREAD] children reset
[THREAD] Terminating thread
[THREAD] I'm a new thread generated by signal 12
[THREAD] Sending message to children
[THREAD] children reset
[THREAD] Terminating thread
```