semctl	sigaddset	pthread_mutex_init	pthread_setaffinity_np	open
sem_init	sigaction	pthread_mutex_lock	pthread_getaffinity_np	fclose
sem_post		pthread_mutex_unlock	sched_getaffinity	read
sem_wait	pthread_join		sched_setaffinity	write
	pthread_exit	malloc	sched_getscheduler	fopen
exit	wait	atoi	sched_setscheduler	exec

mmap - Nel caso in cui non voglio mappare un file, va aggiunto ai vari flag MAP_ANONYMOUS, -1 a file descriptor e 0 ad offset -> **munmap** alla fine
sbrk - Anche se mi sposto solamente di uno, viene comunque allocata un'intera pagina di memoria, come mostrato a lezione nell'esempio del professore
lseek - Utile per sapere la dimensione di un file, in quanto mi basta posizionare il puntatore per l'accesso alla fine e stampare la posizione
fseek - Come sopra ma se lavoro con file e non con canali
getline - Per far fare tutto al sistema, devo passare il puntatore ad un indirizzo NULL e come dimensioni del buffer devo dare 0, in questo modo la memoria verrà allocata automaticamente dal sistema
fcntl - Studiare bene il man per vedere quali argomenti vanno passati
sprintf - Utilizzo la sintassi comoda della printf per scrivere qualcosa in un certo formato all'interno di un buffer di memoria
strok - La prima chiamata, a cui passo il buffer dove ho la stringa ed il delimitatore per i token, mi restituisce il primo token. Alle successive chiamate, invece del primo parametro devo passare il puntatore a NULL, e devo controllare che il token che ricevo non sia uguale a NULL in quanto in quel caso devo fermarmi
fork - Quello che dichiaro prima della chiamata posso accedere come valori e tutto, solamente che se modifico modifico solamente per me. Comunicazione con altri deve avvenire tramite mmap MAP_SHARED, quindi devo ricordare di passare il puntatore alle funzioni nel caso in cui dovessero scrivere qualcosa, magari sincronizzando con semafori/mutex
pthread_create - Il nome della funzione da cui far partire l'esecuzione del thread deve essere un puntatore a Void, gli argomenti li passo anche tramite puntatore a Void

Liste di thread - **pthread_t thread[NUM_THREAD]** | Utilizzare le #define per rendere più portabile e ordinato il codice

Puntatore a puntatori
char **nome_puntatore | nome_puntatore rappresenta l'indirizzo «base» del mega puntatore. *nome_puntatore rappresenta l'indirizzo dello 0-esimo elemento; *(nome_puntatore+1) quello del 1° elemento e così via.

Può essere definito in questo modo anche - **char *nome_puntatore[NUMERO_ELEMENTI]** | nome_puntatore è sempre l'indirizzo base del mega puntatore, agli altri elementi accedo con nome_puntatore[0] (in questo modo accedo sempre agli indirizzi, da deferenziare per accedere all'elemento vero e proprio se serve).

Puntatori - Fare attenzione a quando bisogna utilizzare malloc per evitare segmentation fault vari. Dopo malloc, usare sempre free.

Fflush stdin - Visto che non esiste in libreria, nel caso in cui si usa la scanf va definita come: #define fflush(stdin) while(getchar() != '\n') (Punto e virgola o meno a discrezione, basta che dopo non ci si dimentica della scelta)