université PARIS DIDEROT PARIS 7

Rapport Programmation système

M1 Informatique

Réalisations

Nous avons implémenté le sujet minimal ainsi que les extensions suivantes :

- Détection d'erreur
- lecture et écriture par flots
- Lecture et écriture non bloquantes
- comparaison avec les tubes et les sockets

De plus nous avons aussi implémenté les exemples d'utilisations suivants :

- Un programme de transfert de fichier.
- Un algorithme qui utilise le conduit avec des fork, où les fils se signalent au processus parent.

Description du programme

Notre structure est composée de plusieurs éléments, d'une chaîne de caractères représentant le nom, d'un verrou et d'une condition, d'une adresse de début de conduit , de deux size_t un pour l'atomicité et l'autre pour la taille et enfin de deux int permettant de savoir où en sont les lectures et les écritures. Nous avons fais le choix d'un buffer circulaire car cela nous semblait plus efficace que d'effacer sans cesse ce que nous écrivions et de toujours revenir au début.

algorithme de conduct_create :

Cette fonction permet la création de conduit. La fonction s'articule ainsi :

- Si on veut créer un conduit nommé alors on vérifie tout d'abord si le fichier existe déjà dans le système de fichier, si c'est le cas on abandonne la création en initialisant errno.

  Si le fichier n'existe pas, on le crée et on le mmap.

- Si on veut créer un conduit anonyme, on mmap un conduit déclaré au préalable.

Dans tous les cas, on initialise après le mmap les champs du conduit.

algorithme de conduct_open :

Cette fonction permet l'ouverture d'un conduit déjà existant. On ouvre le fichier (en vérifiant qu'il existe) puis on obtient sa taille pour mmaper le fichier.

algorithme de conduct_close :

Cette fonction permet de fermer le conduit sans le détruire. On commence donc par faire un msync puis on supprime le mmaping avec munmap.

algorithme de conduct_destroy :

Cette fonction nous permet de détruire le conduit. On commence par détruire le variable de condition et le mutex puis on synchronise avec le ficher avant de le supprimer et de supprimer le mmaping avec munmap.

algorithme de conduct_read :

On commence par prendre le verrou du conduit, on calcule ensuite la capacité de lecture sur le conduit.On vérifie par la suite qu'il n'y ait pas plus rien à lire et un caractères de fin de fichier, ensuite si le nombre de caractères à lire respecte l'atomicité nous attendons que le nombre de caractères souhaité soit disponible dans le conduit en attendant un signal d'une écriture , sinon on lit au mieux selon la place disponible(pouvant aller de 0 au nombre de caractères souhaité.Après lecture on décale le compteur de lecture, si l'on a fait un tour du conduit on revient à 0 et on lit le nombre de caractères soustrait au nombre de caractères que l'on vient de lire.Pour finir on signale qu'on a fini de lire et on relâche le verrou.

algorithme de conduct_write :

De même on commence par prendre le verrou.On vérifie ensuite qu'il n'y ait pas un caractères de fin de fichier.On calcule ensuite le nombre de caractères que l'on peut écrire dans le conduit.Si celui-ci est insuffisant et que le nombre de caractères que l'on veut écrire est atomique on attend un signal d'une lecture.Sinon on écrit au maximum de la capacité.Enfin quand le nombre de caractères est suffisant on écrit dans le conduit.Après écriture on décale le compteur d'écriture, si on a fait un tour du conduit on revient à 0 et on écrit le nombre de caractères restant. Pour finir on signale que l'on a fini d'écrire et on relâche le verrou.

algorithme de conduct_write_eof :

On commence par prendre le verrou puis on met à 1 le champs de conduit qui indique la fin de fichier.

algorithme de conduct_readv :

Cette fonction permet de lire le contenu du conduit en stockant chaque octet lu dans la structure iovec.

On commence donc par calculer l'espace dont on dispose dans la structure donnée en argument, puis on lit (en appelant à conduit_read) octet par octet en stockant dans la structure iovec tant qu'on a de la place.

algorithme de conduct_writev :

Cette fonction permet d'écrire le contenu de la structure iovec dans le conduit.

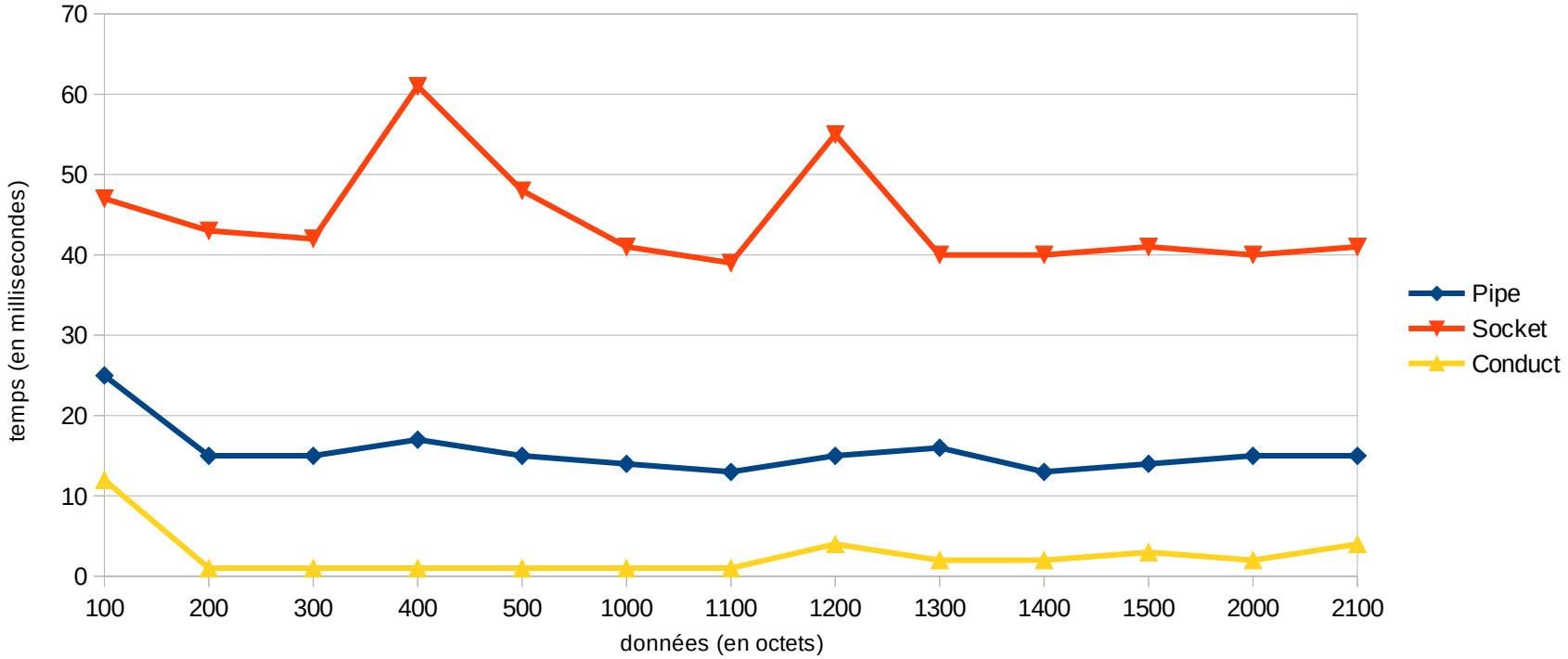
On commence donc par calculer le remplissage de la structure donnée en argument, puis on écrit octet par octet dans le conduit en appelant à conduit_write.

algorithme des écritures et lectures non bloquantes:

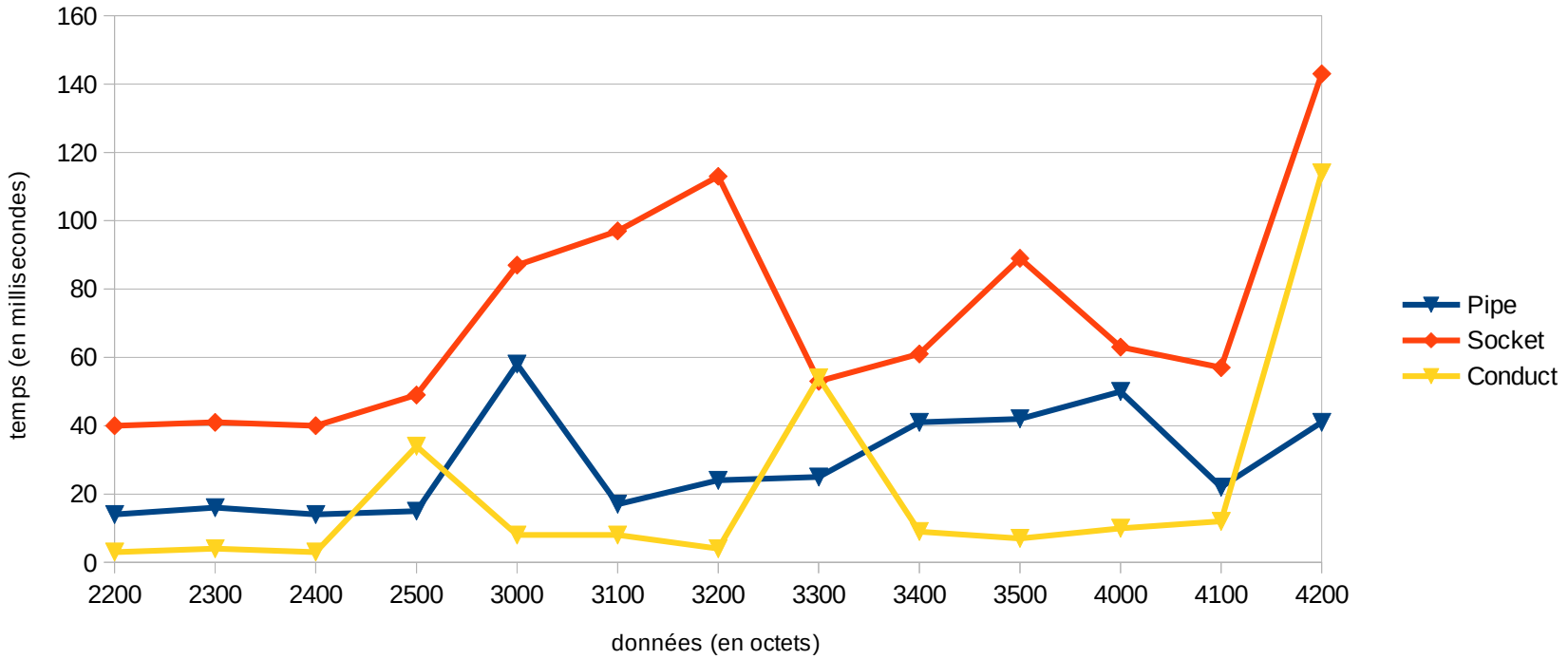
ils sont équivalents aux écritures et lectures non bloquantes à l'exception que si le verrou n'est pas libre ou qu'on doit le bloquer on renvoie -1 et on met errno à EWOULDBLOCK.

Statistiques obtenues

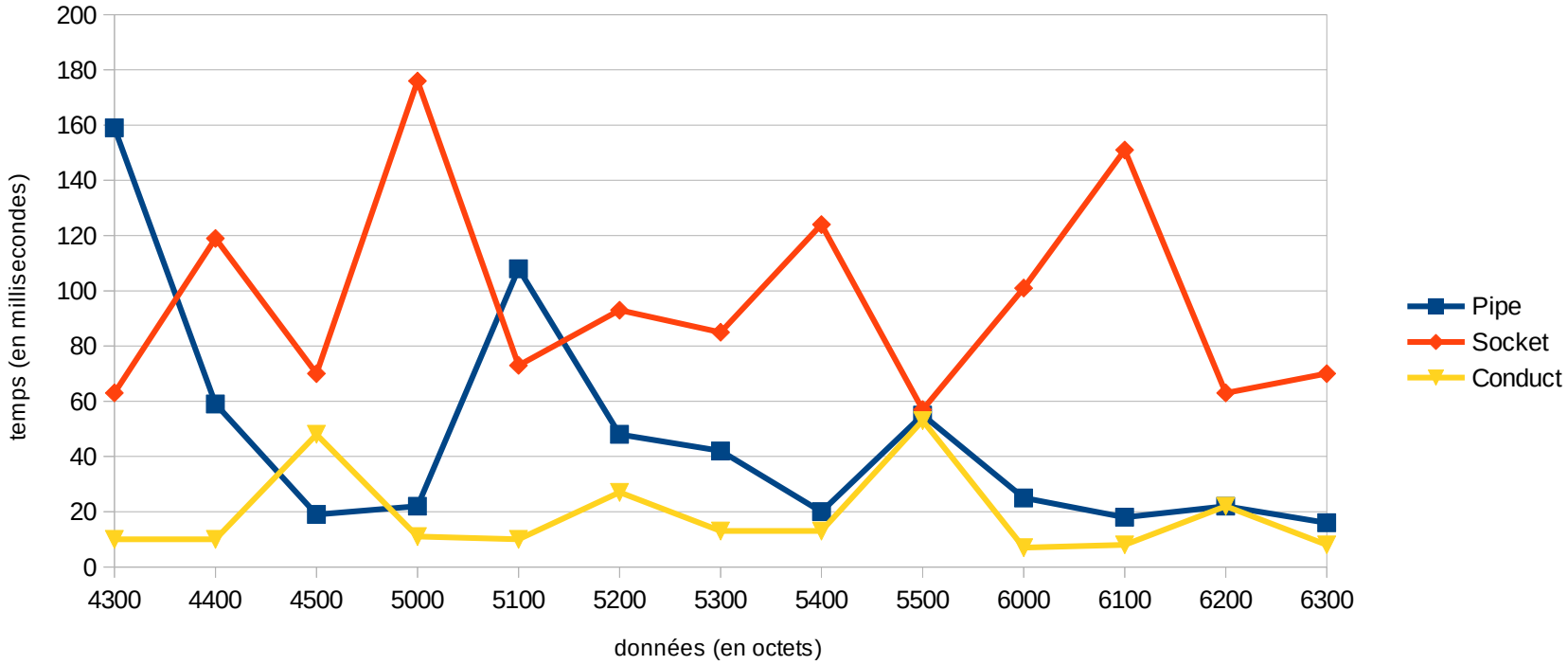
	100	200	300	400	500	1000	1100	1200	1300	1400	1500	2000	2100
Pipe	25	15	15	17	15	14	13	15	16	13	14	15	15
Socket	47	43	42	61	48	41	39	55	40	40	41	40	41
Conduct	12	1	1	1	1	1	1	4	2	2	3	2	4
pipe/conduct	52,00 %	93,33 %	93,33 %	94,12 %	93,33 %	92,86 %	92,31 %	73,33 %	87,50 %	84,62 %	78,57 %	86,67 %	73,33 %
socket/conduct	74,47 %	97,67 %	97,62 %	98,36 %	97,92 %	97,56 %	97,44 %	92,73 %	95,00 %	95,00 %	92,68 %	95,00 %	90,24 %



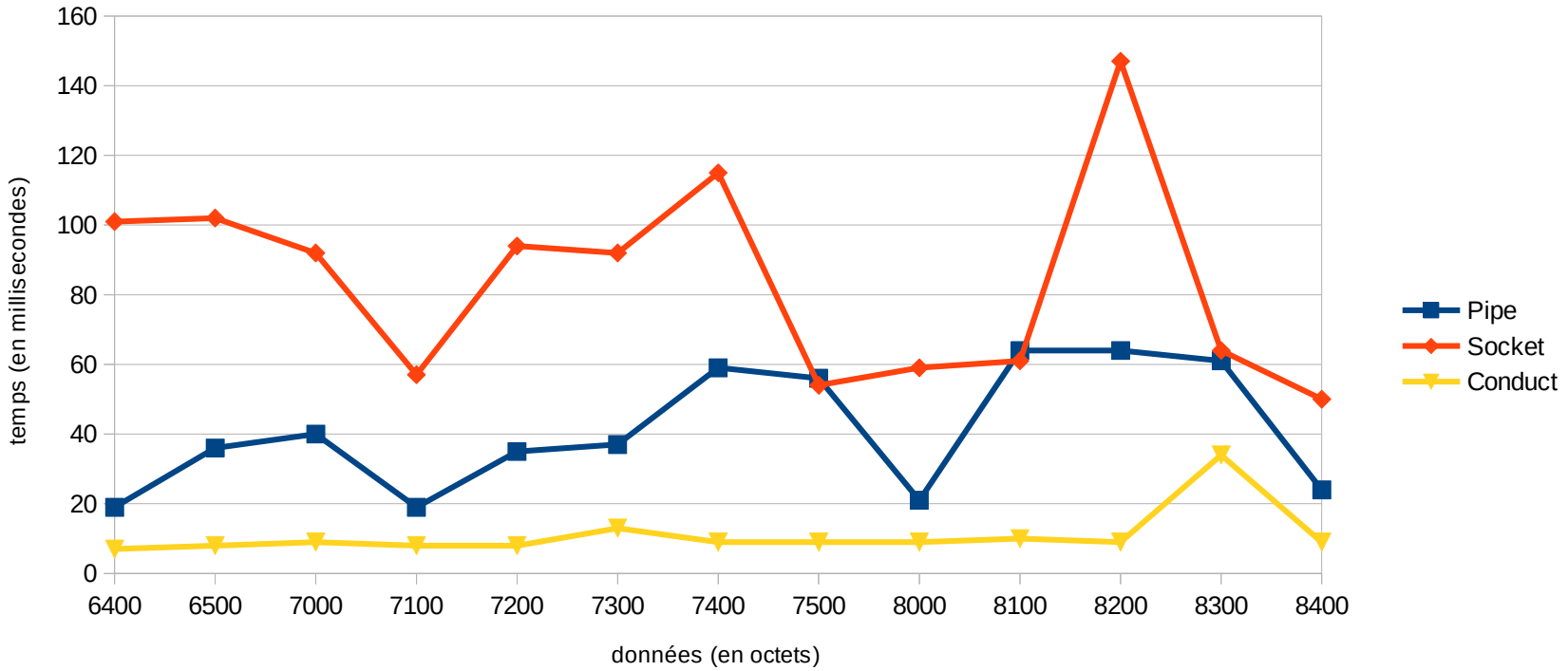
	2200	2300	2400	2500	3000	3100	3200	3300	3400	3500	4000	4100	4200
Pipe	14	16	14	15	58	17	24	25	41	42	50	22	41
Socket	40	41	40	49	87	97	113	53	61	89	63	57	143
Conduct	3	4	3	34	8	8	4	54	9	7	10	12	114
pipe/conduct	78,57 %	75,00 %	78,57 %	-126,67 %	86,21 %	52,94 %	83,33 %	-116,00 %	78,05 %	83,33 %	80,00 %	45,45 %	-178,05 %
socket/conduct	92,50 %	90,24 %	92,50 %	30,61 %	90,80 %	91,75 %	96,46 %	-1,89 %	85,25 %	92,13 %	84,13 %	78,95 %	20,28 %



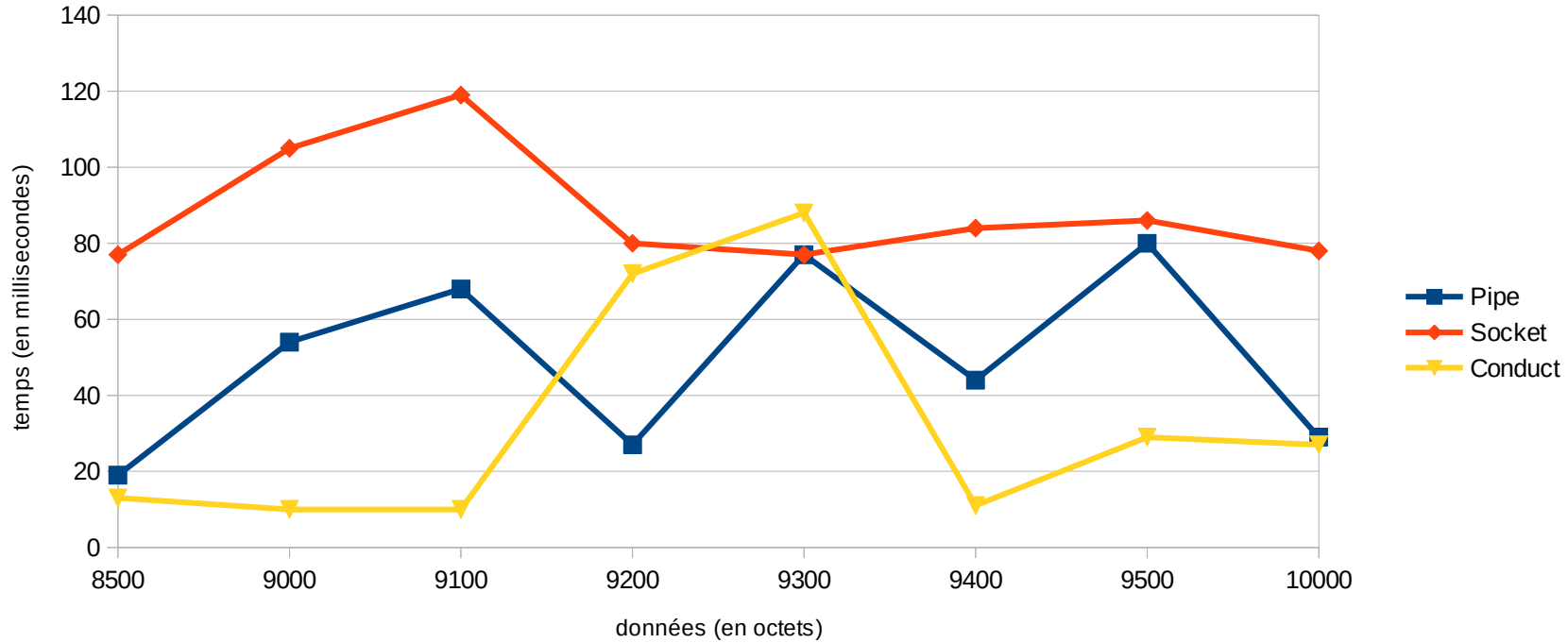
	4300	4400	4500	5000	5100	5200	5300	5400	5500	6000	6100	6200	6300
Pipe	159	59	19	22	108	48	42	20	55	25	18	22	16
Socket	63	119	70	176	73	93	85	124	57	101	151	63	70
Conduct	10	10	48	11	10	27	13	13	53	7	8	22	8
pipe/conduct	93,71 %	83,05 %	-152,63 %	50,00 %	90,74 %	43,75 %	69,05 %	35,00 %	3,64 %	72,00 %	55,56 %	0,00 %	50,00 %
socket/conduct	84,13 %	91,60 %	31,43 %	93,75 %	86,30 %	70,97 %	84,71 %	89,52 %	7,02 %	93,07 %	94,70 %	65,08 %	88,57 %



	6400	6500	7000	7100	7200	7300	7400	7500	8000	8100	8200	8300	8400
Pipe	19	36	40	19	35	37	59	56	21	64	64	61	24
Socket	101	102	92	57	94	92	115	54	59	61	147	64	50
Conduct	7	8	9	8	8	13	9	9	9	10	9	34	9
pipe/conduct	63,16 %	77,78 %	77,50 %	57,89 %	77,14 %	64,86 %	84,75 %	83,93 %	57,14 %	84,38 %	85,94 %	44,26 %	62,50 %
socket/conduct	93,07 %	92,16 %	90,22 %	85,96 %	91,49 %	85,87 %	92,17 %	83,33 %	84,75 %	83,61 %	93,88 %	46,88 %	82,00 %



	8500	9000	9100	9200	9300	9400	9500	10000
Pipe	19	54	68	27	77	44	80	29
Socket	77	105	119	80	77	84	86	78
Conduct	13	10	10	72	88	11	29	27
pipe/conduct	31,58 %	81,48 %	85,29 %	-166,67 %	-14,29 %	75,00 %	63,75 %	6,90 %
socket/conduct	83,12 %	90,48 %	91,60 %	10,00 %	-14,29 %	86,90 %	66,28 %	65,38 %



Temps total pipe	2127
Temps total conduct	947
Temps total socket	4550
Gain pipe/conduct	49,90 %
Gain socket/conduct	78,85 %